우리식조작체계문고

JSP와 Servlet

교육성교육정보쎈터
주체97(2008)년

차 례

머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《우리는 정보기술, 나노기술, 생물공학을 발전시키는데 선차적으로 힘을 넣어야 하며 그중에서도 정보기술, 특히 프로그람기술을 빨리 발전시켜야 합니다.》

오늘 망을 통한 정보의 교류가 급격히 발전하고 사회경제생활에서 그 의의가 날을 따라 커지고있다. 인터네트의 보급범위가 더욱 넓어지고 국가적인 정보고속도로가 필수적인 경제하부구조로 구축되고있는 현실은 콤퓨터들사이의 망통신과 망봉사를 효과적으로 실현하기 위한 능률적인 프로그람들을 많이 개발할 절박한 과제를 제기하고있다.

Servlet와 JSP(Java Server Page)는 모두 봉사기/의뢰기방식의 망봉사를 실현해주는 봉사기측 Java언어, 스크립트언어들이다. 최근 Servlet와 JSP언어는 망통신, 망봉사, 홈페지작성 등 여러 분야에 널리 응용되고있으며 이에 대한 연구가 활발하게 벌어지고있다.

이 책에서는 Servlet와 JSP의 개념과 동작원리, Tomcat와 MySQL봉사기의 설치와 환경설정 방법, 그리고 Servlet와 JSP프로그람작성방법, 대화접속과 쿠키에 대하여 서술하였다. 또한 JSP에 기초한 망열람자계수기, 전자게시판, 대화실, 망연단 등 홈페지작성과 관련한 구체적인 응용실례들을 주었다.

봉사기/의뢰기방식의 망프로그람작성과 관련한 도서로서는 처음으로 집필한것만큼 이 책에 부족점이 많으리라고 본다.

독자들의 좋은 의견을 많이 받아 앞으로 보다 완성할것을 약속하면서 아울러 나라의 정보과학기술을 발전시키기 위한 독자들의 학습과 과학연구사업에서 이 책이 다소나마 도움이 되기를 바란다.

제1장. JSP와 Servlet의 초보

이 장에서는 JSP와 Servlet의 개념과 개발환경, 그리고 프로그람작성방법들을 서술한다.

제1절. Java Server Page의 개념

1.1.1. Java Server Page란

Java Server Page(JSP)는 동적인 웨브페지를 쉽게 만들수 있는 방법을 제공하며 웨브응용프로그람을 간단히 작성, 처리할수 있게 해주는 봉사기측 스크립트언어이다. JSP는 기존의 단순한 HTML를 봉사하던 웨브봉사기의 기능을 보다 발전시켜 웨브기반의 프로그람을 작성할수 있도록 해준다. 또한 Servlet를 기반으로 하고있고 Servlet의 프로그람적인 요소들을 발전시켜 사용자가 쉽게 다룰수 있게 해준다. JSP와 Servlet의 관계를 그림 1-1에서 보여준다.

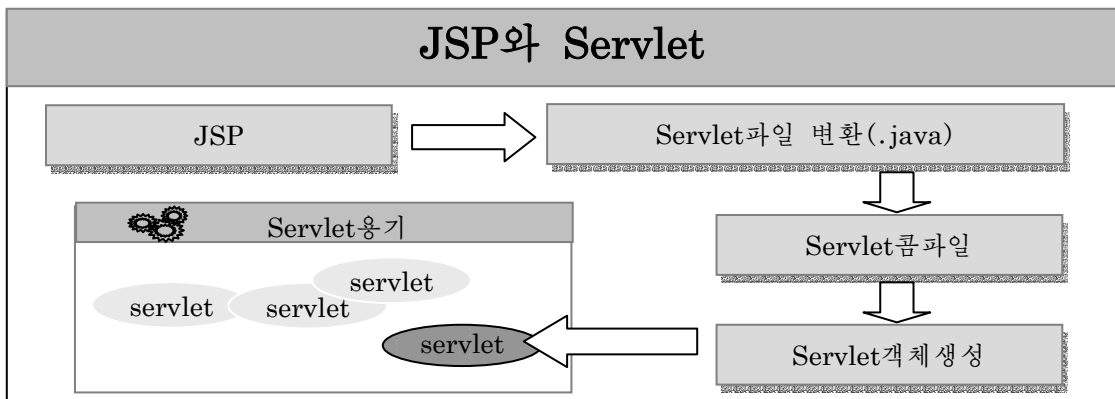



그림 1-1. JSP와 Servlet사이의 관계

JSP는 스크립트언어이므로 작성할 때에만 스크립트의 형태로 만들어지고 사용될 때에는 Servlet파일로 변환,콤파일된다. 다음에 Servlet객체를 생성하여 Servlet Container(Servlet용기)에서 관리한다. 결국 JSP는 Servlet로 되여야 실행될수 있다. 이런 의미에서 《JSP는 Servlet이고 Servlet는 JSP이다.》라고 말할수 있다.

1.1.2. HTML과 JSP의 차이점

- HTML은 단순히 웨브봉사기와 의뢰기사이의 자료 요청과 봉사로 이루어져있다.
- JSP는 HTML보다 한단계 더 나아가 봉사기측프로그람을 추가할수 있다.

그림 1-2를 통하여 HTML봉사와 JSP봉사방식을 알아보기로 하자. JSP봉사인 경우

페지를 요청할 때 먼저 웨브봉사기가 요청을 받고 Servlet와 JSP준위에서 처리를 한다. 그리고 그 결과를 다시 웨브봉사기에 전달하며 의뢰기는 처리결과를 화면에 현시한다.

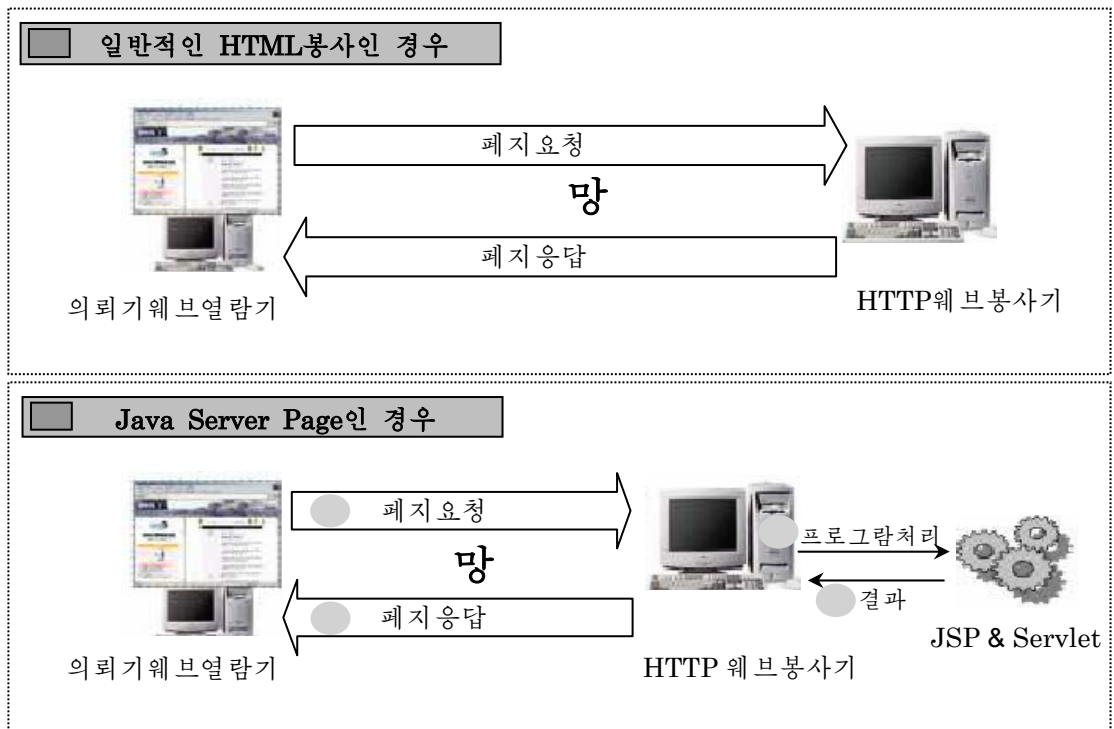



그림 1-2. html과 JSP의 봉사방식

이 봉사방식은 CGI(Common Gateway Interface:공통관문대면부)방식이며 Java용 공통관문대면부가 바로 Servlet나 JSP이다. CGI모형은 사용자의 요청이 있을 때마다 프로세스를 하나씩 생성한다. 그러나 CGI의 계승모형인 Servlet와 JSP에서는 이러한 결함을 개선하여 확장된 CGI모형인 내부프로쎄스방식을 사용하고있다. 내부프로쎄스방식에서는 기억기에 서고를 한번 적재시킨 프로쎄스를 여러 요청들에 리용할수 있다. 또한 Java언어를 그대로 활용할수 있으므로 Java의 우점을 모두 가지고있다. 그리고 토막처리를 한다는것이 CGI와 JSP가 구별되는 차이점이다.

제2절. Servlet와 JSP

1.2.1. Servlet와 JSP의 관계

Servlet는 봉사기에서 실행되는 프로그람으로서 봉사기에서 프로그람을 처리하고 그 결과를 의뢰기에 전송하는 역할을 한다. JSP 역시 Servlet와 같은 처리를 한다. 그러나 Servlet는 JSP의 상위에 있다고 할수 있다. Servlet가 Java프로그람언어준위에서 처리한다면 JSP는 스크립트준위에서 처리한다. JSP는 내부적으로 Servlet를 사용하고있으며 스크립트페지는 종국에는 Servlet로 콤파일되여 봉사요청에 응답한다.(그림 1-3)

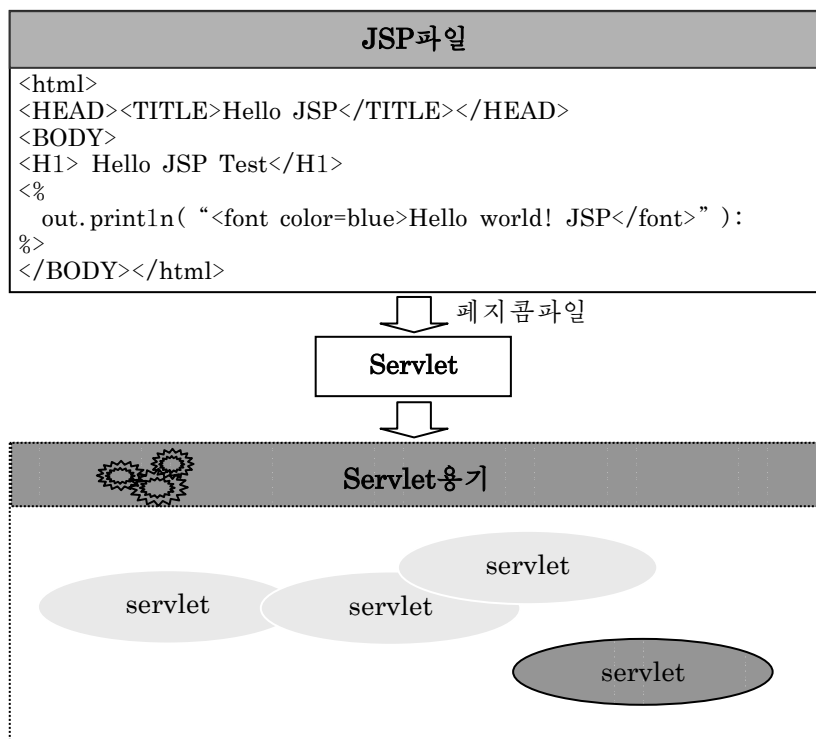



그림 1-3. JSP가 Servlet로 변환되는 과정

그림 1-3은 JSP페지가 봉사될 때 Servlet로 콤파일되는 과정을 보여주고있다. Servlet는 Servlet용기에 삽입되여 봉사된다. Servlet는 하나의 프로세스를 공유하는 방식으로 되여있으므로 한번만 콤파일하여 Servlet용기에 삽입시키면 다음부터는 Servlet용기에 생성된 Servlet객체를 공유하여 사용하면 된다. 결국 JSP는 스크립트로 쉽게 프로그람을 작성하기 위한것이다.

1.2.2. JSP가 콤파일되여 Servlet용기에 적재되는 시점

JSP파일을 만들어 봉사를 하는 등록부에 넣어두면 나머지는 Java Server Page내부에서 알아서 처리한다.

- JSP페지가 처음 요청을 받은 경우에는 물론 .class로 콤파일되고 다음에 Servlet용기에 적재되여 봉사된다.
- 의뢰기에서 JSP를 요청할 때 먼저 해당한 JSP의 Servlet객체가 Servlet용기에 있는가 없는가를 검사하여 만일 있다면 직접 실행에 들어간다. 그러나 없으면 다시 Servlet생성과정을 거쳐야 한다.
- Java Server Page는 수정검사프로그람으로 JSP파일이 수정되였는가를 검사한 후 수정되였으면 jsp파일을 class로 콤파일하여 Servlet용기에 다시 적재한다.(그림 1-4)

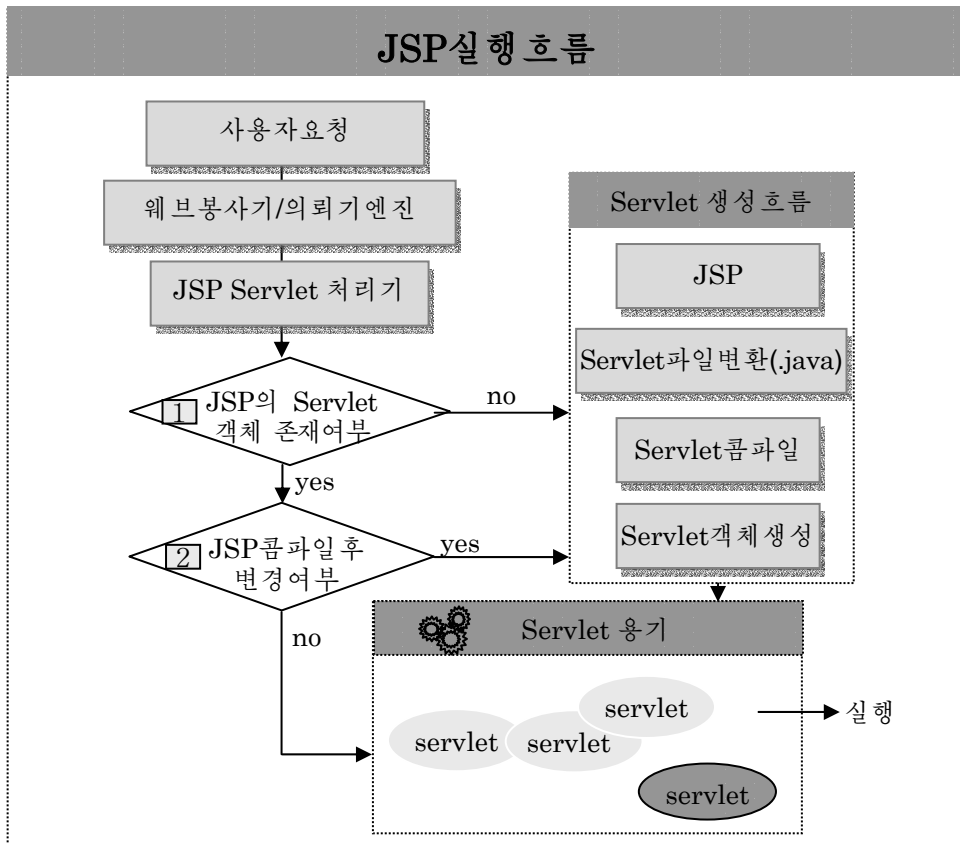



그림 1-4. JSP의 실행흐름

그림에서 보는바와 같이 JSP의 실행은 먼저 웨브봉사기나 Servlet엔진에서 의뢰기의 요청을 받는다. 봉사기에서 요청을 받으면 JSP처리기 즉 JSP용기에서의 처리를 거쳐 Servlet 객체의 존재여부와 파일변경여부를 확인한다. 변경된 사항이 있으면 새로 원천파일을 생성하고 콤파일과정을 거치며 이미 적재되여있지만 변경되지 않았으면 현재 존재하는것을 리용한다. 적재하지 않았을 경우 처음부터 적재를 시작하여 원천파일을 만들고 콤파일과정을 거쳐 새롭게 실행한다.

제3절. JSP와 Servlet의 개발환경

여기에서는 JSP를 실행하기 위하여 Windows XP환경에서 Tomcat4.0.4의 사용방법에 대하여 설명한다.

1.3.1. Tomcat의 설치와 환경설정

- Tomcat4.0.4 내리적재

그림 1-5. Tomcat의 내리적재

Java거점으로 가서 http://java.sun.com/products/jsp/download.html을 찾으면 Tomcat를 내리적재받을수 있는 항목이 나타난다. 이것을 찰칵하면 Apache홈페지로 안내해준다. 또는 http://jakarta.apache.org로 찾아가도 된다.

- 환경설정

System Properties
System Restore
Automatic Updates
Remote
General
Computer Name
Hardware
Advanced
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings
User Profiles
Desktop settings related to your logon
Settings
Startup and Recovery
System startup, system failure, and debugging information
Settings
Environment Variables
Error Reporting
OK
Cancel
Apply

Environment Variables
User variables for kangchol
Variable
Value
classpath
c:\j2sdk1.4.1_01\lib\tools.jar;c:\Tomcat...
Path
C:\MATLABR11\bin;c:\j2sdk1.4.1_01\bin
TEMP
C:\Documents and Settings\kangchol\Lo...
TMP
C:\Documents and Settings\kangchol\Lo...
TOMCAT_HOME
c:\Tomcat 4.0
New
Edit
Delete
System variables
Variable
Value
classpath
c:\tomcat 4.0\lib
ComSpec
C:\WINDOWS\system32\cmd.exe
JAVA_HOME
c:\j2sdk1.4.1_01
NUMBER_OF_P...
1
OS
Windows_NT
New
Edit
Delete
OK
Cancel

그림 1-6. Tomcat봉사기의 환경설정

My Computer에서 마우스오른쪽단추를 찰칵하고 Properties항목을 선택하면 System Properties대화칸이 나타나는데 여기서 Advanced표쪽을 찰칵한다. 이때 나타나는 대화칸에서 Envirnoment Variables단추를 찰칵하면 Envirnoment Variables대화칸이 나타난다. 그림 1-6과 같이 환경변수를 설정한다. 실례로 Envirnoment Variables대화칸에서 New단추를 찰칵하여 그림과 같이 환경변수 TOMCAT_HOME을 추가한다.

이렇게 하면 모든 환경설정이 끝난다. Tomcat 4.0이상판본에서는 TOMCAT_HOME 대신에 CATALINA_HOME을 입력해도 된다.

1.3.2. Tomcat봉사기의 실행과 확인

Tomcat봉사기를 실행하기 위해서는 Tomcat의 bin등록부에 있는 start.bat를 실행하거나 DOS창에서 start.bat를 입력한 후 Enter건을 누르면 된다. 아래의 그림은 Tomcat봉사기를 실행한 화면이다.

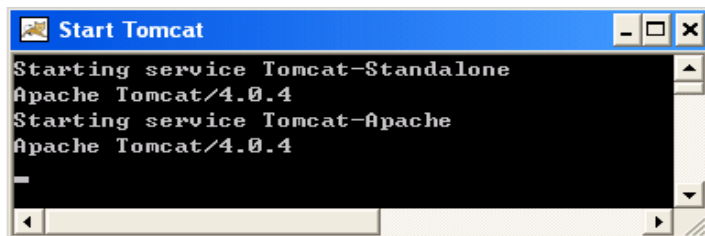



그림 1-7. Tomcat의 시동화면

Tomcat봉사기를 실행하려면 TOMCAT_HOME과 JAVA_HOME을 설정하면 된다. start.bat파일을 실행한 후 정확히 설치되였는가를 확인하기 위해서는 웨브에서 http://localhost:8080을 입력하여 Tomcat가 동작하는가를 확인하여야 한다.

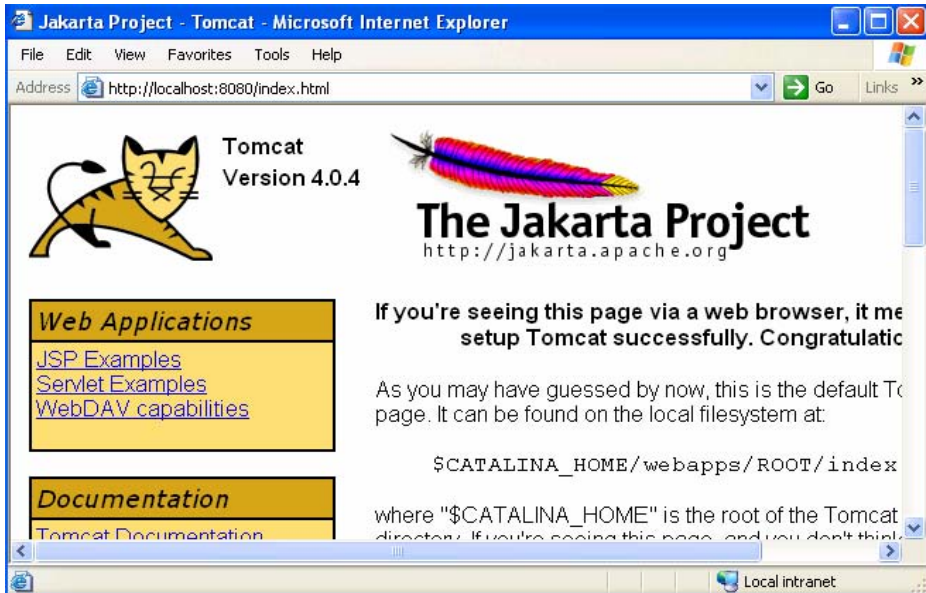



그림 1-8. Tomcat봉사기의 홈페지

우의 그림과 같이 나타나면 정확히 동작하는것이다.

또한 classpath를 사용하는 경우 servlets.jar파일을 classpath에 반드시 추가하여야 한다. 만일 servlets.jar파일이 classpath에 지정되여있지 않으면 Tomcat의 실행에서는 문제가 발생하지 않지만 Tomcat의 Bean을 콤파일할 때 classpath오유를 발생시킨다.

1.3.3. Linux조작체계에서의 Tomcat설치와 실행

Linux조작체계에서의 Tomcat설치와 실행도 Windows에서와 류사하다.

거점 http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/

에서 Linux가동기반의 jakarta-tomcat-4.0-YYYYMMDD.zip파일을 사용자가 설치하려는 등록부안에 풀어서 넣는다. 그리고 Tomcat가 설치된 등록부의 경로에 환경변수 TOMCAT_HOME 또는 CATALINA_HOME을 설정한 다음 아래의 지령을 실행하여 Tomcat 봉사기의 실행을 확인한다.

```
$CATALINA_HOME/bin/startup.sh
```

또는

```
cd $CATALINA_HOME/bin
./startup.sh
```

Tomcat를 기동한 다음 웨브열람기에서 다음의 지령을 실행하여 봉사기가 정확히 동작하는가를 확인한다.

```
http://localhost:8080/
```

이때 그림 1-8과 같은 화면이 현시되면 Tomcat가 오유없이 동작하는것으로 된다.

봉사기를 닫을 때에는 아래와 같이 한다. 즉

```
$CATALINA_HOME/bin/shutdown.sh
```

또는

```
cd $CATALINA_HOME/bin
./shutdown.sh
```

제4절. 웨브응용프로그람작성

1.4.1. 웨브응용프로그람의 작성절차

응용프로그람을 작성하기 위해서는 다음과 같이 하여야 한다.

▶ 먼저 webapps등록부에 아래의 그림과 같이 MySample등록부를 만든다.

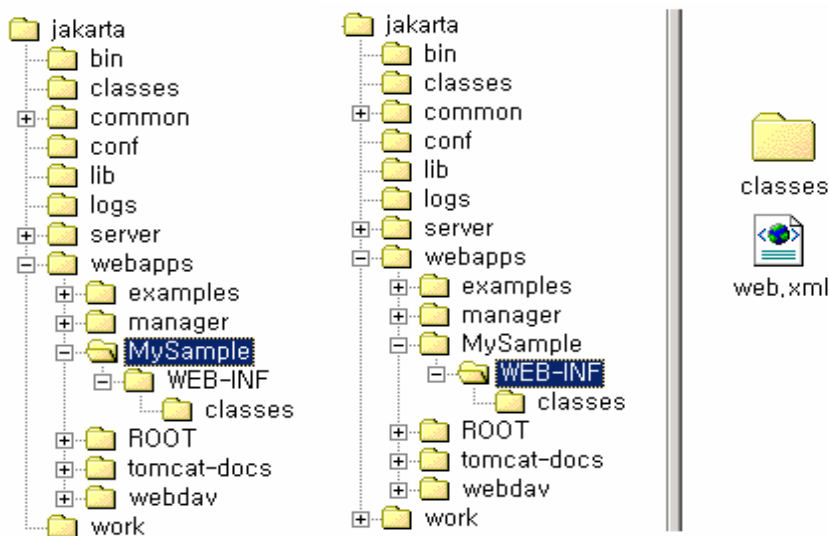



그림 1-9. 응용프로그람실행을 위한 등록부

▶ 그 아래에 WEB-INF등록부를 만들고 다시 그 아래에 classes라는 등록부를 만든다.

▶ ROOT등록부의 web.xml파일을 MySample등록부의 WEB-INF부분등록부에 복사한다. 복사가 끝나면 MySample등록부안에 간단한 HTML파일을 만들어 http://localhost:8080/ MySample/로 접근해 본다.

실례 1-1

Hello.html
<HTML><BODY> <H1>Hello! My Sample Web Application</H1> </BODY></HTML>
MySample등록부에 hello.html을 삽입한다.

이때 Tomcat봉사기는 실행상태여야 한다. 아래의 그림(그림 1-10)은 웨브봉사기에 접근하여 특정한 파일을 실행하는 실례를 보여주고있다.

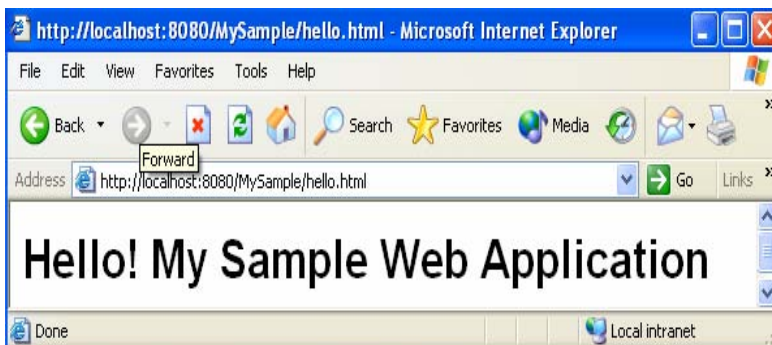
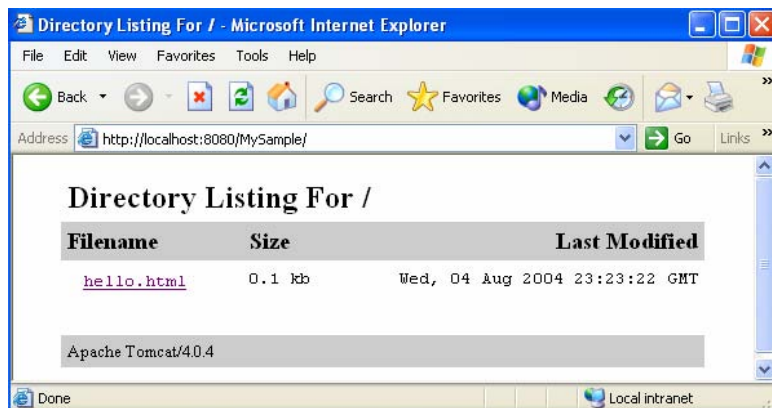



그림 1-10. 실례 1-1의 실행결과

작성된 프로그람은 MySample등록부안에 있어야 한다. WEB-INF등록부안에 있는 HTML이나 jsp파일에는 접근할수 없다. WEB-INF등록부안에는 웨브응용프로그람에서 사용하는 class파일들이 놓인다.

1.4.2. Hello World! Servlet의 작성

Hello World! Servlet를 작성해보자. 아래와 같이 작성하여 MySample/WEB-INF/classes등록부에 HelloServlet.java파일을 만든다. 그리고 classes등록부에서 콤파일한다.

실례 1-2.

HelloServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                         throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.println(" <HTML><BODY>");
      out.println("<H1> Hello World </H1>");
      out.println("</BODY></HTML>");
  }
}
```

콤파일방법은 다음과 같다.

```
C:\Tomcat\webapps\MySample\WEB-INF\classes>javac  HelloServlet.java
C:\Tomcat\webapps\MySample\WEB-INF\classes>dir
2007-10-11   02:01a                    689  HelloServlet.class
2007-10-11   01:47a                    430  HelloServlet.java
```

이 실례의 실행결과는 그림 1-11과 같다.

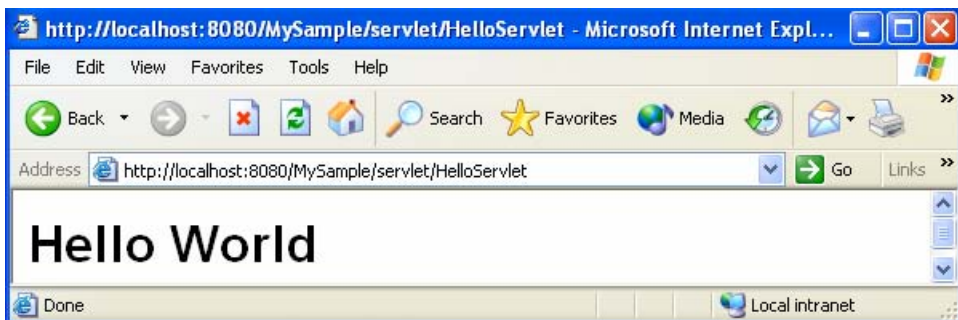



그림 1-11. 실례 1-2의 실행결과

프로그람설명

- 우선 javax.servlet.*, javax.servlet.http.* 패키지들을 적재한다.
- HttpServlet를 계승한다.
- doGet메쏘드를 재정의한다.
- PrintWriter out = response.getWriter(); response객체로부터 흐름을 얻는다.
- 흐름을 리용하여 의뢰기에 자료를 전송한다.

콤파일이 끝나면 아래의 주소로 Servlet에 접근해본다.

http://localhost:8080/MySample/servlet/HelloServlet

- 주의할것은 HelloServlet앞에 servlet라는 수식어가 붙는다는것이다.

1.4.3. Package형태의 Hello World! Servlet작성

앞에서 학습한 Hello World! Servlet를 패키지형태로 작성해보자. Hello Wolrd! Servlet와 기본적으로 같고 다만 패키지형태일 때 Servlet에 접근하는 방법에서 차이가 있다. 아래의 원천코드파일(HelloPackageServlet.java)을 classes등록부안에 넣는다.

파일을 패키지형태로 만들기 위하여 패키지열쇠어를 리용한다. 즉 원천코드에 package org.jabook.myhello;을 삽입한다.

classes등록부에 java파일을 넣고 -d를 추가하여 패키지파일을 콤파일한다. -d를 추가하면 패키지등록부까지 자동적으로 생성해준다.

주의하여야 할것은 접근할 때 다음과 같이 해당한 패키지명을 전부 사용해야 한다는것이다.

http://localhost:8080/MySample/servlet/org.jabook.myhello.HelloPackageServlet

아래의 그림(그림 1-12)에서 보다싶이 패키지가 등록부형식으로 생성되였지만 패키지형태의 호출로서 접근하여야 한다.

그림 1-12. 콤파일에서 패키지의 생성과정

실례 1-3

HelloPackageServlet.java

```
package org.jabook.myhello;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloPackageServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println(" <HTML><BODY>");
        out.println("<H1> Hello World Package Servlet </H1>");
        out.println("</BODY></HTML>");
    }
}
```

콤파일은 아래와 같이 한다.

C:\Tomcat\webapps\MySample\WEB-INF\classes>javac -d . HelloPackageServlet.java

콤파일하면 classes등록부아래에 org, jabook, myhello등록부들이 순차적으로 생성된다. 실행결과는 그림 1-13과 같다.

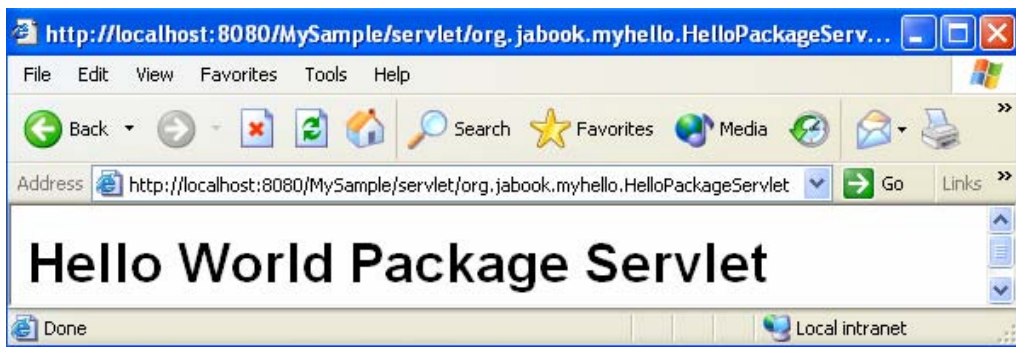



그림 1-13. 실례 1-3의 실행결과

1.4.4. Hello World! JSP의 작성

가장 간단한 Hello World! JSP프로그람을 작성해보자. 아래의 실례는 HTML코드로 된 HTML파일을 확장자만 .jsp로 붙인것이다. 그러나 이 파일은 JSP로 동작한다.

실례 1-4

hello.jsp

```
<HTML><HEAD><TITLE>Hello JSP</ TITLE></ HEAD ><BODY>
<H1> Hello JSP Test</H1>
<%

   out.println("<font color=blue>Hello World! JSP</font>");
%>
</ BODY></ HTML>
```

접근할 때에는 다음과 같이 한다.

http://localhost:8080/MySample/hello.jsp

Servlet처럼 콤파일이나 등록부의 복잡한 절차없이 바로 우의 주소로 접근하면 Hello JSP를 볼수 있다.(그림 1-14) 조선글은 깨져서 나타나게 된다. 조선글을 사용하자면 다음과 같은 코드를 맨 웃줄에 삽입하여야 한다.

```
<%@ page contentType= "text/html; charset=big5" %>
```

charset를 지정하지 않으면 조선글을 볼수가 없다.

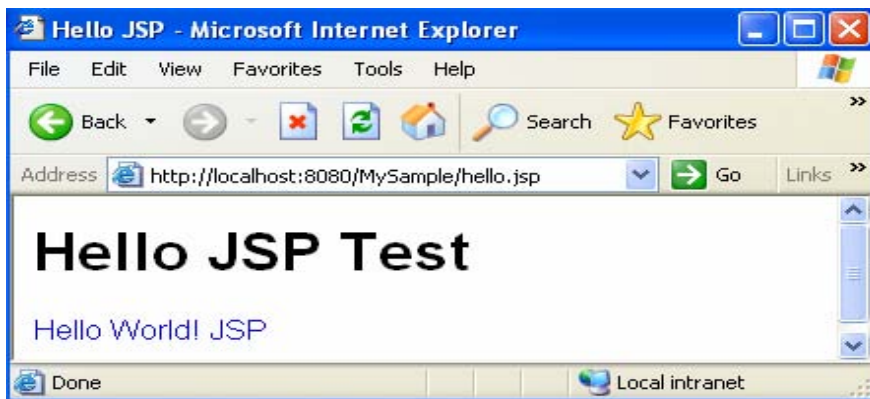



그림 1-14. 실례 1-4의 실행결과

1.4.5. JSP에서 Bean의 사용

JSP파일에서 Bean을 사용하는 방법에 대하여 보기로 하자. JSP에서 Bean을 사용하기 위해서는 먼저 Bean을 작성하여야 한다. statement를 비공개(private)성원마당으로 선언하고 공개메쏘드(public method)인 set와 get를 작성한다. 현재 Bean은 패키지형태로 되여있으며 classes등록부에서 생성된 후 콤파일할 때 –d추가선택을 리용하면 자동으로 패키지등록부가 생성된다.

실례 1-5

HelloBean.java

package hello;
public class HelloBean {
  private String statement=" ";
  public String getStatement(){
    return statement;
  }
  public void setStatement(String statement){
    this.statement = statement;
  }
}

C:\Tomcat\webapps\MySample\WEB-INF\classes>javac -d . HelloBean.java

아래와 같이 Bean을 생성한 후 JSP에서 Bean을 적재하여 사용한다.(그림 1-15)

즉 %@page import="hello.*" %

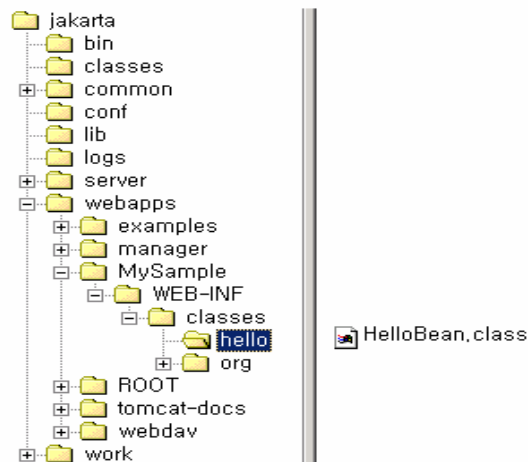



그림 1-15. 실례 1-5에서의 파일등록부위치

아래의 코드는 Bean을 사용하여 입력한 글을 출력하는 실례이다.

실례 1-6

hellobean.jsp
```
<%@page import="hello.*" %>
<HTML><BODY><BR>
<%
  String str=null;
  HelloBean myhello = new HelloBean();
  myhello.setStatement("JSP Bean Test!!");
  str = myhello.getStatement();
  out.println("<font size=5 color=blue>"+str+"</font>");
%>
<BR></ BODY></HTML>
```

Bean을 사용하는 절차

- 패키지형태의 Bean을 작성한다.
- JSP에서 사용할 Bean패키지를 적재한다.
- new로 Bean을 생성하여 일반 Java클라스처럼 사용한다.

실행결과는 그림 1-16과 같다.

그림 1-16. 실례 1-6의 실행결과

1.4.6. JSP Bean꼬리표의 리용

JSP파일내에서 Bean을 사용하는 다른 하나의 방법은 JSP액션꼬리표의 useBean꼬리표를 리용하는 방법이다. 액션꼬리표를 통한 Bean의 적재방법에 대하여 보기로 하자.

useBean액션꼬리표의 일반적인 형식은 아래와 같다.

```
<jsp:useBean id= "myhello" class= "HelloWorld" scope= "page" />
```

useBean꼬리표는 새로운 객체를 생성하거나 이미 생성된 객체의 참조를 얻는데 리용

한다. 우의 명령문은 jsp:useBean끄리표를 리용하여 HelloWorld라는 Bean클라스의 객체를 생성하고있다. 객체의 이름은 myhello이다. 이렇게 정의된 myhello라는 Bean객체는 일반적인 객체와 똑같은 방법으로 메쏘드를 호출한다. 형식에서 scope= "page" 는 Bean이 생성된 페지안에서만 유효하다는것을 의미한다.

그러면 useBean끄리표를 사용하여 《JSP useBean Test!!》를 출력하는 실례를 보기로 하자. 여기서 Bean파일은 우에서 만든 HelloBean파일을 그대로 사용한다.

실례 1-7

HelloUseBean.jsp

```
<jsp:useBean id="myhello" class="hello.HelloBean" scope="page"/>
<HTML><BODY><BR>
<%
  String str = null;
  myhello.setStatement("JSP useBean Test!!");
  str = myhello.getStatement();
  out.println("<font size=5 color=blue>"+str+"</font>");
%>
<BR></BODY></HTML>
```

그림 1-17. 실례 1-7의 실행결과

실행된 화면(그림 1-17)을 보면 우에서 new를 통해 Java Bean을 생성한것과 별로 차이가 없는것을 알수 있다. 그러나 JSP코드를 보면 간단해졌다.

Java Bean을 리용하는데서 useBean끄리표의 사용방법

- id에는 생성하려는 객체의 이름을 입력한다. 여기에서는 myhello라는 객체를 생성하고있다.
- class에는 패키지이름까지 모두 서술한 클라스의 이름을 넣는다.
- scope에서는 Bean이 존재하는 범위를 정한다. 여기서는 page를 설정함으로써 Bean을 생성한 페지안에서만 사용하게 하고있다.

<jsp:useBean id= "myhello" class= "Hello.HelloBean" scope= "page" />

id=myhello를 통하여 Bean으로 문자렬을 전송하고 다시 불러들이는 등 Java클라스처럼 사용할수 있다.

제2장. Servlet의 기초

이 장에서는 Servlet의 개념과 기본계승구조에 대하여 서술한다.

제1절. Servlet의 개념

Servlet는 Applet와 비슷한 방식으로 동작한다. Applet는 Applet클라스를 계승받아 메쏘드를 재정의하여 사용하며 Servlet는 HttpServlet를 계승받아 메쏘드를 재정의하여 사용한다. Applet를 만들 때 Applet를 계승받고 init, start, stop, destory 등의 메쏘드를 재정의하여 여기에 여러가지 기능을 부여한다. 아래의 그림에서 Applet를 계승받아 사용하는 주요 성원메쏘드들을 보여주었다.

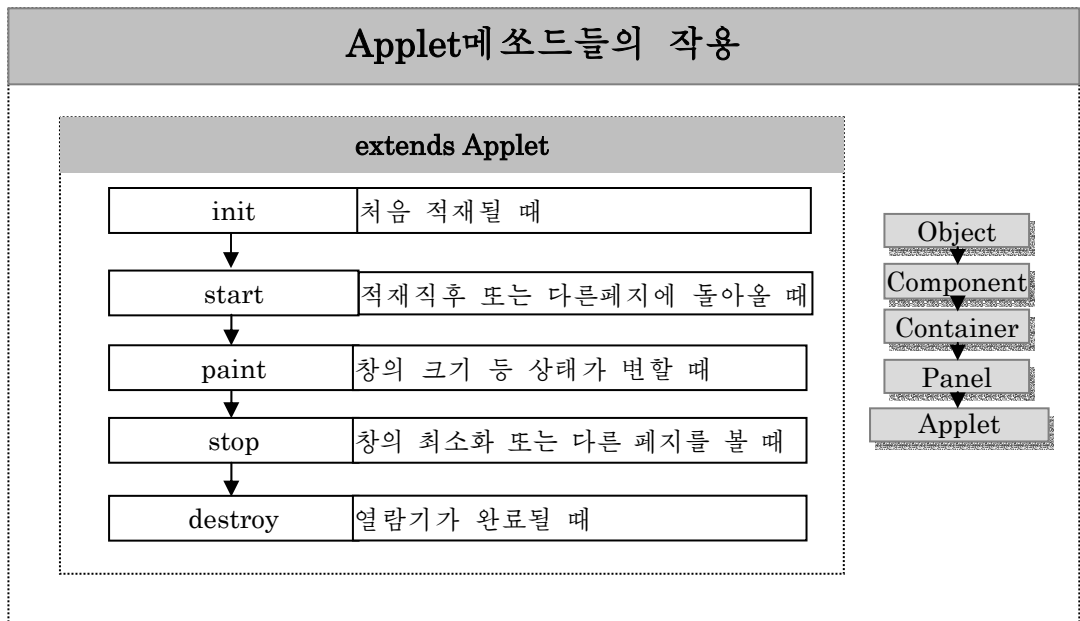



그림 2-1. Applet메쏘드들

Servlet는 Servlet대면을 실현하여 GenericServlet를 만들고 이것을 다시 Http규약에 맞게 확장한 HttpServlet클라스를 계승한 후 내부메쏘드를 재정의하여 사용한다. 그림 2-2에서는 HttpServlet를 계승한 후 재정의하여 사용할수 있는 주요 성원메쏘드 service, doGet, doPost, doHEAD들을 보여주고있다.

Applet처럼 Servlet의 생명주기도 크게 3개의 메쏘드로 이루어져있다. 즉 init, service, destroy메쏘드이다. init와 destroy메쏘드는 Servlet가 생성, 파괴될 때

사용되는 메쏘드이며 service메쏘드는 의뢰기의 요청에 응답하는 메쏘드이다. 그리고 service 메쏘드는 의뢰기의 요청방식에 따라 doGet, doPost를 자동으로 호출한다.

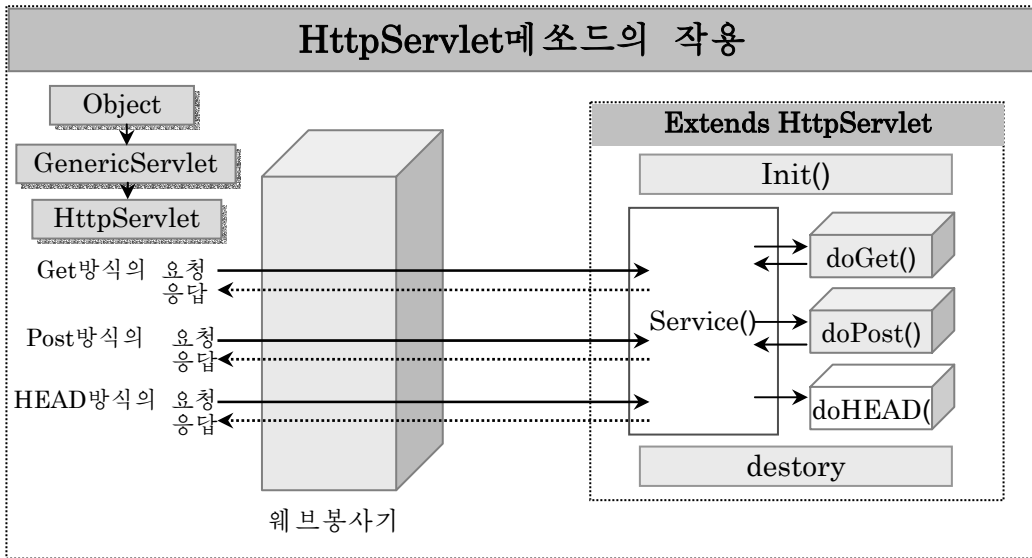



그림 2-2. HttpServlet의 내부메쏘드들

Applet는 의뢰기에서 동작하는 프로그람이지만 Servlet는 봉사기에서 동작하는 프로그람이다. 단지 웨브환경에서 이루어지는 요청을 봉사기에서 처리하기 위한 프로그람적인 수단으로 사용된다.

제2절. Servlet의 동작과정

Servlet는 의뢰기의 요청이 있을 때 그것을 처리하고 만일 자료기지에 접속하여야 하는 경우에는 Servlet에서 접속과 관련한 업무처리를 한 다음 그 결과를 의뢰기에 되돌려 보낸다.(그림 2-3)

Servlet의 동작과정은 다음과 같다.

① 의뢰기의 요청
② Servlet처리기의 8080포구에서 요청접수
③ Servlet용기에서 해당 Servlet검색
④ 해당 Servlet가 자료기지에 대한 작업을 해야 하는 경우 DB에 련결
⑤ 모든 작업이 완료되면 응답으로 결과를 되돌려준다.

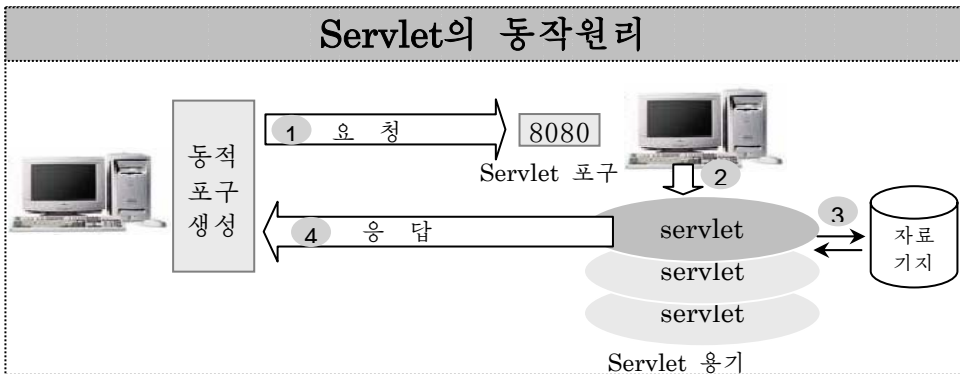



그림 2-3. Servlet의 동작과정

Servlet는 Servlet용기가 관리하며 매개 Servlet는 Servlet클라스를 적재하여 하나의 객체를 생성한다. 그리고 하나의 객체기억기를 공유하면서 Servlet객체의 service메쏘드만으로 의뢰기의 요청을 처리한다. 매개 Servlet는 HttpServlet클라스를 계승받아 service, doGet, doPost 등의 메쏘드를 재정의하게 되며 의뢰기의 요청에 따라 자동으로 요청에 적합한 메쏘드를 호출한다.

제3절. Servlet의 계승구조

Servlet를 만들 때 보통 HttpServlet를 계승한다. 그것은 대부분 Http규약상에서 작업을 하기때문에 Http규약에 맞게 확장된 HttpServlet클라스를 계승하여 재정의한다. HttpServlet는 여러개의 대면을 포함한 GenericServlet를 계승한다.(그림 2-4)

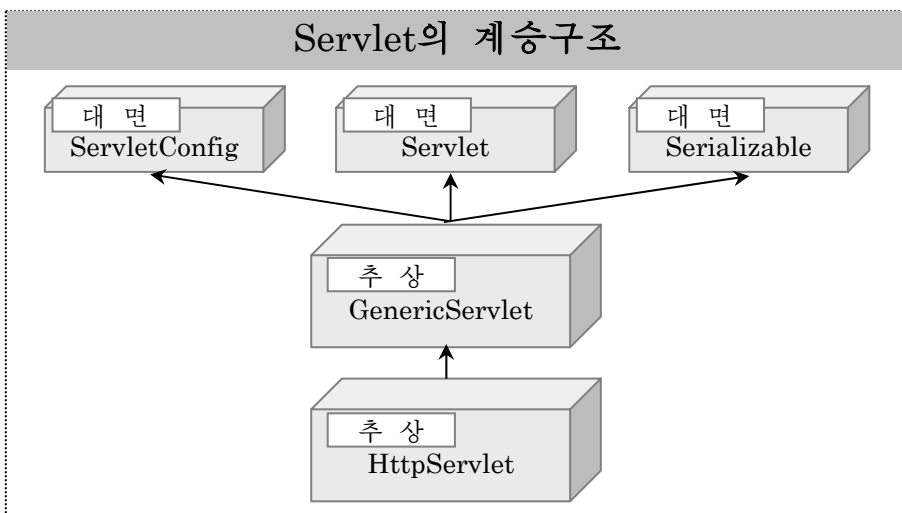



그림 2-4. Servlet의 계승구조

HttpServlet클라스의 상위에는 GenericServlet가 있다. 이것은 Servlet로 의뢰기와 망통신을 할 때 필요한 대면을 실현한다. 그리고 실현된 GenericServlet를 Http규약에 맞게 다시 확장시킨다. 이것이 바로 HttpServlet이다. HttpServlet는 Http규약에 맞게 만들어진 클라스이므로 Http규약을 사용하는 경우에는 HttpServlet를 사용하면 된다.

2.3.1. GenericServlet추상클라스

GenericServlet는 3개의 대면을 실현한다.(그림 2-5) Servlet와 ServletConfig는 Servlet통신에 필요한 메쏘드들을 가지고있으며 GenericServlet에서 모두 실현된다. 그러나 GenericServlet는 추상클라스이므로 그 자체로는 사용할수 없고 반드시 계승해야만 사용할수 있다.

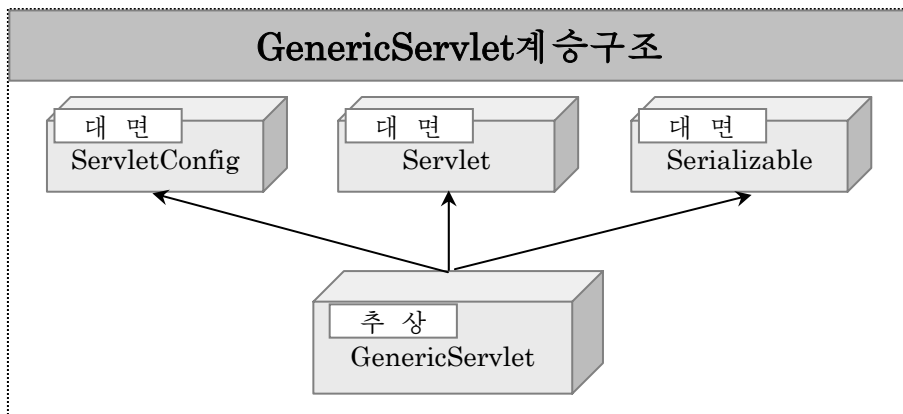



그림 2-5. GenericServlet의 계승구조

GenericServlet의 구성을 보면 다음과 같다.

① ServletConfig대면
② Servlet대면
③ Serializable대면

GenericServlet에서 실현되는 대면들에 포함되여있는 메쏘드들과 GenericServlet자체에서 만들어진 메쏘드들은 일정한 관계를 가지고있다. GenericServlet에서는 Servlet대면에 포함된 메쏘드들의 업무처리를 제공한다. 한편 환경설정이나 초기변수에 관련된 사항들은 ServletConfig가 담당한다.

Http규약이외의 Ftp규약 등에 대응한 Servlet를 작성하는 경우에도 GenericServlet를 계승하여 Servlet를 만들어야 한다.

GenericServlet는 추상클라스이므로 반드시 계승하여 리용하여야 하며 이때 계승받은 클라스에서는 service메쏘드만을 재정의하여 사용하면 된다.

2.3.2. GenericServlet의 실례

다음의 실례는 GenericServlet를 리용하여 URL로부터 통보문을 받아 출력해주는 프로그람이다. 프로그람에서는 GenericServlet클라스를 계승한 다음 service메쏘드를 재정의하여 의뢰기의 요청을 처리한다.

실례 2-1

GenericServletTest.java

```
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;
public class GenericServletTest extends GenericServlet {
   public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
      String message = req.getParameter("message");
      PrintWriter out = res.getWriter();
      out.println("<HTML><BODY>");
      out.println("<H1>" + message + "</H1>");
      out.println("</BODY ></HTML>");
   }
}
```

```
C:\Tomcat\webapps\MySample\WEB-INF\classes>javac GenericServletTest.java
C:\Tomcat\webapps\MySample\WEB-INF\classes>dir GenericServletTest.*
C:\Tomcat\webapps\MySample\WEB-INF\classes
2007-10-19  10:56a   1,081        GenericServletTest.class
2007-10-19  10:56a   529          GenericServletTest.java
```

주소칸에 message=Hello,GenericServlet를 입력하여 실행한다.

그림 2-6. 실례 2-1의 실행결과

콤파일이 완료되면 아래의 주소로 Servlet에 접근할수 있다.

http://localhost:8080/MySample/servlet/GenericServletTest?message=Hello!GenericServlet

프로그람설명

여기서는 웨브열람기의 주소칸을 통하여 의뢰기의 요청이 이루어진다. 웨브열람기에서 그림 2-6과 같이 주소칸에 입력하면 GenericServlet의 service메쏘드가 자동으로 호출되며 이때 req와 res는 자동으로 Servlet에서 service를 호출하여 참조파라메터형태로 넘어온다.

service메쏘드의 참조파라메터 HttpServletRequest Request와 HttpServletResponse Response는 자동적으로 넘어오며 이 두 참조파라메터는 의뢰기의 요청과 응답을 처리한다. 실례에서는 URL을 통하여 참조파라메터가 넘어올 때 이것을 참조파라메터인 req로부터 얻어낸다.

String message = req.getParameter("message");

그리고 의뢰기에 응답을 보내기 위한 흐름을 얻기 위하여 res객체의 getWriter메쏘드를 사용한다.

PrintWriter out = res.getWriter();

다음 의뢰기와 련결된 흐름 out를 리용하여 의뢰기로 자료를 전송한다. 여기서는 out객체의 println메쏘드를 리용하여 의뢰기로 자료를 보내고있다.

표 2-1에서 GenericServlet클라스의 메쏘드들을 소개한다.

표 2-1. GenericServlet클라스의 메쏘드들

메 쏘 드	설 명
void init()	Servlet의 초기화를 진행한다.
void init(ServletConfig config)	Servlet를 호출하여 필요한 자원을 할당하는 등 Servlet초기화를 진행하고 봉사를 시작할수 있도록 한다.
abstract void service(ServletRequest req, ServletResponse res)	Servlet가 초기화된 후 의뢰기로부터 온 요청에 대한 봉사를 진행한다. 이 메쏘드는 추상메쏘드이므로 Servlet를 작성하기 위해 GenericServlet클라스를 계승하는 모든 하위클라스에서 반드시 실현해주어야 하는 메쏘드이다.
void destroy()	Servlet를 완료하려는 경우 이것을 호출하여 필요한 자원을 할당, 해제하는 등의 작업을 할수 있다.
java.lang.String getInitParameter(java.lang.String name)	주어진 이름의 초기화참조파라메터의 값을 포함하고있는 문자렬을 얻는다.
java.util.Enumeration getInitParameterNames()	Servlet를 위한 초기화참조파라메터의 이름을 enumeration형으로 얻는다.
ServletConfig getServletConfig()	Servlet에 대한 초기화나 시작 참조파라메터 등을 포함하고있는 ServletConfig객체를 얻는다.

메 쏘 드	설 명
ServletContext getServletContext()	Servlet가 실행되는 Servlet에 대한 정보를 포함하고있는 ServletContext객체를 얻는다.
java.lang.String getServletInfo()	Servlet의 작성자(author), 판본(version), 저작권(copyright) 등과 같은 Servlet관련정보를 얻기 위하여 재정의해주어야 한다.
void log(java.lang.String msg)	Servlet의 클라스이름과 Servlet례외통보문을 Servlet의 등록파일에 기록한다.
void log(java.lang.String message, java.lang.Throwable t)	체계례외통보문을 Servlet등록파일에 기록한다.

제4절. Servlet의 대면

Servlet프로그람은 init, service, destroy인 3가지 메쏘드에 의하여 초기화, 봉사, 파괴의 과정을 거친다는것을 우에서 언급하였다. 그림 2-7에서 보는바와 같이 Servlet가 적재될 때 init메쏘드를 한번만 호출하며 의뢰기의 요청이 있을 때마다 service메쏘드를 반복적으로 호출하게 된다. 이 3가지 메쏘드는 Servlet대면의 가장 기본적인 메쏘드이며 Servlet의 생명주기와 관련된 메쏘드들이다.

2.4.1. Servlet대면

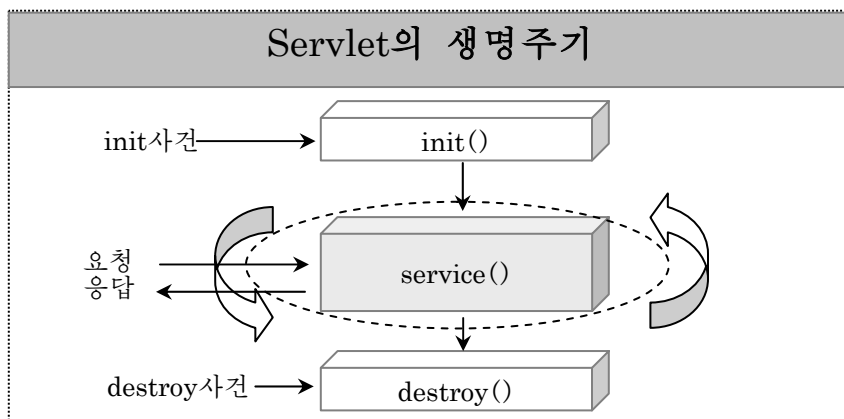



그림 2-7. Servlet의 생명주기

Servlet에 요청이 처음으로 들어왔을 때 해당 Servlet는 Servlet용기에 의해 자동으로 기억기에 적재된다. 기억기에 Servlet클라스가 적재된 다음 객체를 생성하며 객체의 생성과 동시에 init메쏘드를 호출한다. init메쏘드는 Servlet적재시 한번만 호출되며 오유가 발

생했을 때 UnavailableException이나 ServletException을 발생시킨다.

Servlet객체는 기억기를 해제하는 경우 destroy메쏘드를 호출한다. 만약 성공적으로 init메쏘드가 호출되였다면 service메쏘드를 수행하여 의뢰기의 요청에 반응한다. 두번째 의뢰기의 요청이 있을 때부터는 service메쏘드를 호출하여 의뢰기의 요청에 대응한다. service메쏘드는 의뢰기의 요청방식에 따라 Get방식이면 doGet메쏘드를, Post방식이면 doPost를 호출한다. Servlet객체가 더 이상 봉사를 진행할 필요가 없는 경우 기억기에서 제거하기 위하여 destroy()메쏘드를 호출한다. destroy()메쏘드가 호출되면 페품회수기는 객체의 기억기를 제거한다.

아래의 표 2-2에서는Servlet대면의 메쏘드들을 소개한다.

표 2-2. Servlet대면의 메쏘드들

Servlet대면	
void init(ServletConfig config)	Servlet를 호출하여 필요한 자원을 할당하는 등의 Servlet의 초기화를 진행하고 봉사를 시작할수 있도록 한다.
void service(ServletRequestreq, ServletResponse res)	Servlet가 초기화된 후 의뢰기로부터 온 요청에 대한 봉사를 수행한다.
void destroy()	Servlet를 완료할 때 호출하여 필요한 자원을 할당해제하는 등의 작업을 할수 있도록 한다.
ServletConfig getServletConfig()	Servlet에 대한 초기화나 시작 참조파라메터 등을 포함하고있는 ServletConfig객체를 얻는다.
java.lang.String getServletInfo()	Servlet의 작성자(author), 판본(version), 저작권(copyright) 등과 같은 Servlet관련정보를 제공해줄 때 재정의해주어야 한다.

2.4.2. ServletConfig대면

ServletConfig대면에 포함된 메쏘드들은 Servlet를 초기화하는 동안 해당 Servlet에 정보를 전달하기 위하여 사용된다. 이러한 정보에는 Servlet의 봉사기와 관련된 정보를 나타내는 ServletContext, 이름과 값이 쌍으로 이루어진 초기화참조파라메터 등이 있다.

다음의 실례는 ServletConfig대면을 사용하여 Servlet 및 JSP의 판본정보, 봉사기의 정보 등 Servlet와 관련된 여러 정보들을 출력해주는 프로그람이다.

실례 2-2

```
ConfigTest.java
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;
public class ConfigTest extends GenericServlet {
   public void service(ServletRequest request, ServletResponse response)
         throws ServletException, IOException {
      ServletConfig config = getServletConfig();
      PrintWriter out = response.getWriter();
      out.println("<HTML><BODY><H4>");
      out.println("Server Info:"+config.getServletContext().getServerInfo());
      out.println("<BR><BR>");
      out.println("Context attributes:<BR>");
      ServletContext context = config.getServletContext();
      Enumeration  e = context.getAttributeNames();
         while(e.hasMoreElements()) {
               String key =(String)e.nextElement();
               Object value = context.getAttribute(key);
               out.println("\t" + key + " = " + value + "<BR>");
         }
      out.println("<BR><BR>");
      out.println("Java Server: "+context.getMajorVersion()+"."+context.getMinorVersion());
      out.println("</H4></BODY></HTML>");
   }
}
```

```
C:\Tomcat\webapps\MySample\WEB-INF\classes>javac  ConfigTest.java
C:\Tomcat\webapps\MySample\WEB-INF\classes>dir  ConfigTest.*
C:\Tomcat\webapps\MySample\WEB-INF\classes 등록부
2007-10-20   08:51p                    1,864  ConfigTest.class
2007-10-20   08:51p                    1,058  ConfigTest.java
```

그림 2-8은 이 실례의 실행결과이다.

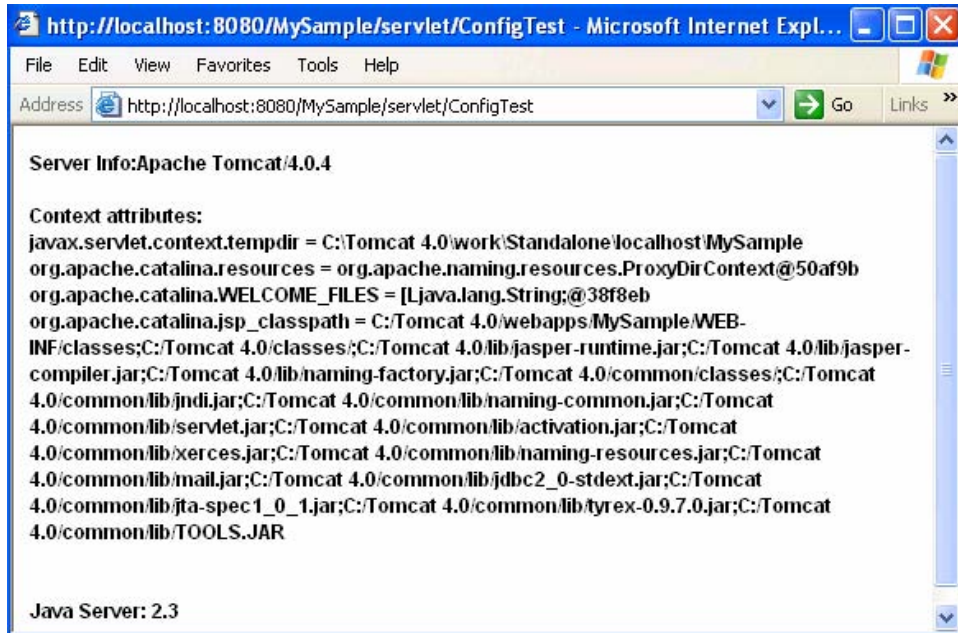



그림 2-8. 실례 2-2의 실행결과

콤파일이 완료되면 아래의 주소로 Servlet에 접근할수 있다.

http://localhost:8080/MySample/servlet/ConfigTest

프로그람설명

Servlet를 실행시키면 봉사기의 정보, 문맥정보, Java봉사기 즉 jsp의 정보 등이 렬거되는것을 확인할수 있다.

ServletConfig객체 config는 getServletConfig()메쏘드를 사용하여 얻는다. 이렇게 얻어진 ServletConfig객체를 리용하여 여러가지의 Servlet정보를 얻을수 있다.

ServletConfig config = getServletConfig();

그리고 의뢰기에 응답을 하기 위한 흐름을 얻기 위하여 response객체로부터 다음과 같은 메쏘드를 사용한다.

PrintWriter out = response.getWriter();

봉사기정보를 알아내여 출력한다. 여기서는 Tomcat 4.0.4가 출력되였다.(그림 2-8)

out.println("Server Info:"+config.getServletContext().getServerInfo());

ServletConfig객체 config를 사용하여 ServletContext객체를 얻는다. 그리고 렬거형(Enumeration)변수를 리용하여 Servlet문맥이 가지고있는 정보들을 출력한다. ServletContext클라스를 리용하여 Servlet의 관련정보 및 웨브응용프로그람의 여러 정보를 알아볼수 있다.

ServletContext context = config.getServletContext();

Enumeration e = context.getAttributeNames();

이렇게 ServletConfig대면은 Servlet의 여러 정보뿐아니라 웨브응용프로그람의 정보를 알아낼수도 있다. 아래에서는 ServletConfig대면의 메쏘드들에 대하여 소개한다.(표 2-3)

표 2-3. ServletConfig대면의 메쏘드들

ServletConfig대면	
java.lang.String getInitParameter(java.lang.String name)	주어진 이름의 초기화참조파라메터의 값을 포함하고있는 문자렬을 얻는다.
java.util.Enumeration getInitParameterNames()	초기화참조파라메터의 이름을 enumeration형으로 얻는다.
ServletContext getServletContext()	Servlet가 실행되는 Servlet엔진에 대한 정보를 포함하고있는 ServletContext객체를 얻는다.
java.lang.String getServletName():	Servlet의 이름을 귀환한다.

제5절. HttpServlet추상클라스

HttpServlet는 일반적으로 의뢰기로부터 온 요청을 받아서 처리하고 그 결과를 다시 의뢰기에게 되돌려주는 작업을 수행한다. HttpServlet는 Servlet프로그람을 작성할 때 가장 많이 사용하는 클라스로서 Servlet를 만들 때 보통 doGet와 doPost를 재정의하여 사용한다. 앞에서 본것처럼 GenericServlet클라스를 계승하였으면 service메쏘드를 재정의하여 의뢰기의 요청을 처리한다. 그러나 HttpServlet는 이 service메쏘드를 재정의하여 만들어졌으므로 일단 의뢰기의 요청이 있으면 HttpServlet클라스의 service메쏘드가 자동적으로 호출된다. 그리고 service메쏘드의 내부에서는 의뢰기의 요청방식에 따라 doGet, doPost메쏘드가 자동적으로 호출된다. 아래에서는 이러한 처리방식을 그림으로 보여주고있다.(그림 2-9)

그림 2-9는 GenericServlet를 계승하여 클라스를 생성하면 service메쏘드를 재정의하고 HttpServlet를 계승하여 Servlet를 만들면 의뢰기의 요청방식에 따라 doGet나 doPost등의 메쏘드를 계승한다는것을 보여주고있다.

HttpServlet는 GenericServlet를 계승하므로 HttpServlet자체의 Servlet생명주기는 그대로 적용된다. Servlet대면에서처럼 init, service, destroy의 생명주기를 가진다. 그러나 HttpServlet는 service메쏘드를 재정의하여 사용하지 않는다. service메쏘드는 자동적으로 호출되며 의뢰기의 요청방식에 따라 doGet나 doPost 등을 재정의하여 사용한다. 실

제상 service메쏘드는 Http규약의 의뢰기요청방식에 따라 각각 다른 메쏘드를 내부에서 호출할수 있다. 아래에서는 의뢰기의 요청방식에 따라 service메쏘드가 호출하는 메쏘드들을 보여주고있다.

① doGet메쏘드: Servlet가 HTTP GET요청을 처리하기 위하여 재정의한다.
② doPost메쏘드: Servlet가 HTTP POST요청을 처리하기 위하여 재정의한다.
③ doPut메쏘드: Servlet가 HTTP PUT요청을 처리하기 위하여 재정의한다.
④ doDelete메쏘드: Servlet가 HTTP DELETE요청을 처리하기 위하여 재정의한다.

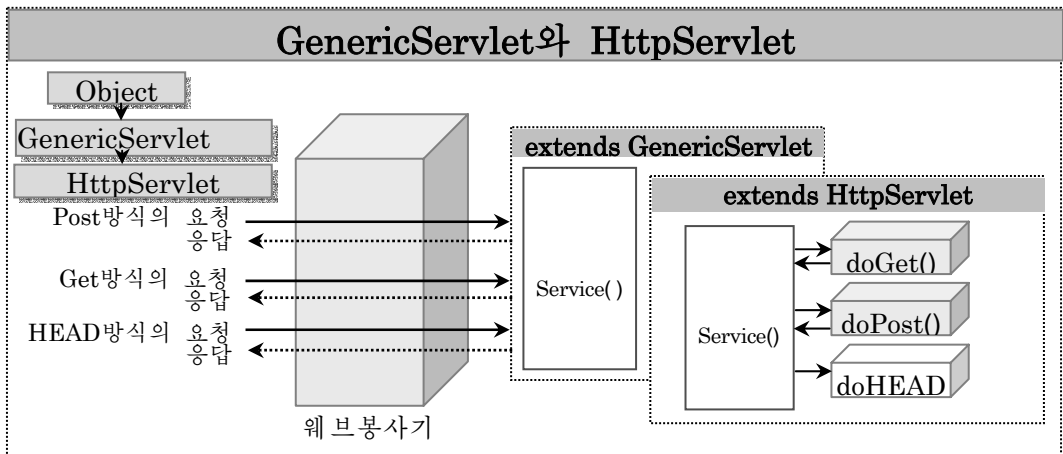



그림 2-9. HttpServlet의 내부메쏘드들

2.5.1. doGet메쏘드

아래의 프로그람은 의뢰기가 Get방식으로 요청할 때 결과를 의뢰기에 보내는 실례이다. 아래와 같이 작성하여 MySample등록부의 WEB-INF의 classes등록부에 HelloDoGet.java파일을 넣고 classes등록부안에서 콤파일을 진행한다.

실례 2-3

HelloDoGet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloDoGet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");
```

```
        out.println("<HTML><BODY>");
        out.println("<H1> Hello doGet Test</H1>");
        if(name != null){
           out.println("<H2>"+name+"</H2>");
        }else{
            out.println("<H2>nothing parameter</H2>");
        }
        out.println("</BODY></HTML>");
    }
}
```

```
C:\Tomcat\webapps\MySample\WEB-INF\classes>javac  HelloDoGet.java
C:\Tomcat\webapps\MySample\WEB-INF\classes>dir  HelloDoGet.*
C:\Tomcat\webapps\MySample\WEB-INF\classes 등록부
2007-10-22   01:46a              1,064  HelloDoGet.class
2007-10-22   01:46a                594  HelloDoGet.java
```

- 주소칸에 자료가 없을 때의 접근

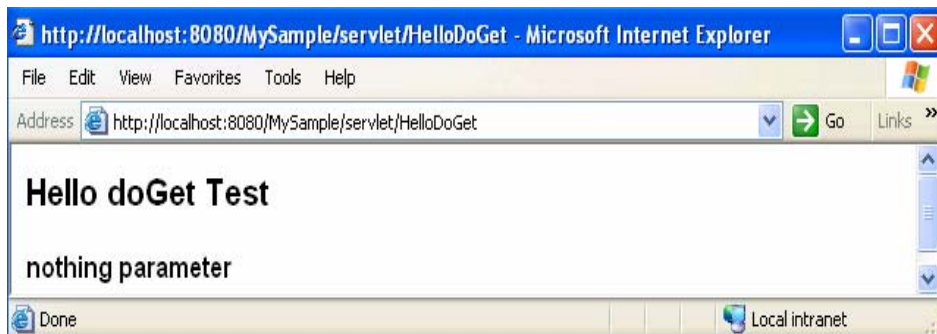



그림 2-10. 실례 2-3의 실행화면

콤파일이 완료되면 아래의 주소로 Servlet에 접근할수 있다.
http://localhost:8080/MySample/servlet/HelloDoGet

- 주소칸에 자료가 있을 때의 접근

그림 2-11. 실례 2-3의 다른 실행화면

콤파일이 완료되였으면 아래의 주소로 Servlet에 접근할수 있다.

http://localhost:8080/MySample/servlet/HelloDoGet?name=Kim Yong Chol

주의할것은 HelloDoGet앞에 servlet라는 수식어가 붙는다는것이다.

프로그람설명

웨브열람기에서 주소로 호출하였을 때 Get방식의 의뢰기요청이 이루어진다. 우의 웨브열람기에서 그림과 같이 주소칸에 입력하면 HelloDoGet의 doGet메쏘드는 자동적으로 호출되며 이때 request와 response는 참조파라메터로 넘어온다. 물론 doGet는 service메쏘드에 의해 자동호출되며 service메쏘드는 의뢰기의 요청방식을 확인하게 된다.

doGet메쏘드의 형식은 다음과 같다.

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

doGet메쏘드의 참조파라메터 HttpServletRequest와 HttpServletResponse는 service메쏘드에 의하여 자동적으로 넘어오며 이 두 참조파라메터는 의뢰기로부터의 요청과 응답을 클라스로 넘긴다. 우의 실례에서는 Get방식으로 의뢰기가 자료를 넘길 때 이것을 참조파라메터인 request로부터 얻어낸다.

```
String name = request.getParameter("name");
```

그리고 의뢰기에 응답하기 위한 흐름을 얻어내기 위하여 response객체로부터 다음과 같은 메쏘드를 사용한다.

```
PrintWriter out = response.getWriter();
```

그리하여 의뢰기와 련결된 흐름 out를 얻어 의뢰기로 자료를 전송하기만하면 된다.

대체로 out객체의 println메쏘드를 리용하여 의뢰기로 자료를 보낸다.

2.5.2. doPost메쏘드

실례 2-4는 의뢰기가 Post방식으로 요청할 때 결과를 의뢰기에게 응답해주는 실례이다. Servlet의 doPost메쏘드의 사용방법은 doGet메쏘드의 사용방법과 거의 같다. 의뢰기가 웨브열람기의 주소칸에서 요청을 했을 때에는 Get방식의 요청이 이루어지지만 HTML의 홈전송방식을 Post로 설정하고 Servlet를 요청했을 때에는 doPost가 호출된다.

그림 2-12은 Form끄리표를 리용하여 Servlet를 요청하는것을 도식화한 그림이다.

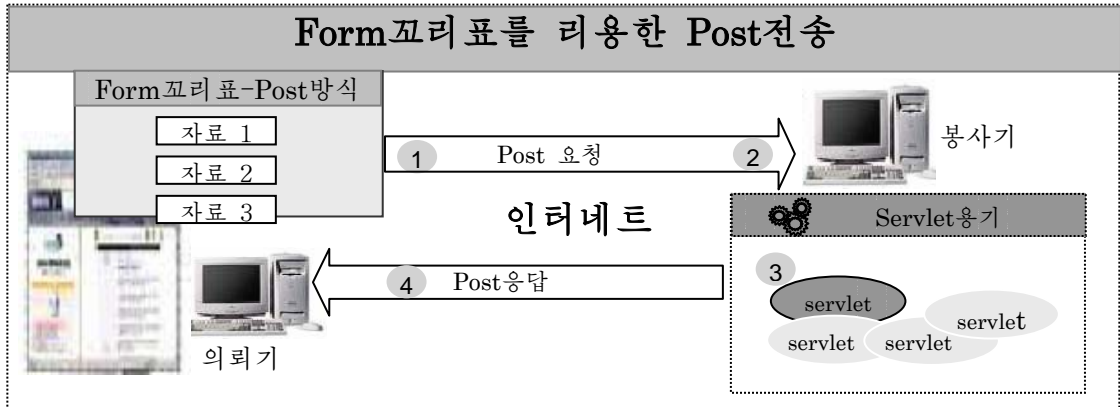



그림 2-12. Form끄리표를 리용한 Post전송방식

원천코드를 작성한 다음 MySample등록부의 WEB-INF의 classes등록부에 HelloDoPost.java파일을 넣고 classes등록부안에서 콤파일한다. 그리고 Post방식으로 전송하기 위하여 HTML의 FORM끄리표에서 method를 Post로 지정한다. 실례를 들어보자.

실례 2-4

HelloDoPost.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloDoPost extends HttpServlet {
   public void doPost(HttpServletRequest request,  HttpServletResponse response)
                  throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      String name = request.getParameter("name");
      out.println("<HTML><BODY>");
      out.println("<H1> Hello  doPost Test</H1>");
      if(name != null){
         out.println("<H2>"+name+"</H2>");
      }else{
```

```
        out.println("<H2>nothing parameter</H2>");
      }
      out.println("</BODY></HTML>");
    }
}
```

HelloDoPost.html

```
<HTML><HEAD><TITLE>doPost 검사</TITLE></HEAD>
<BODY>
<H1>doPost 검사</H1>
<FORM action="/MySample/servlet/HelloDoPost" method="Post">
  <INPUT type="text" name="name">
  <INPUT type="submit" name="submit1">
</FORM>
</BODY>
</HTML>
```

```
C:\Tomcat\webapps\MySample\WEB-INF\classes>javac HelloDoGet.java
C:\Tomcat\webapps\MySample\WEB-INF\classes>dir HelloDoGet.*
C:\Tomcat\webapps\MySample\WEB-INF\classes 등록부
2007-10-21   01:46a              1,064 HelloDoGet.class
2007-10-21   01:46a              594 HelloDoGet.java
```

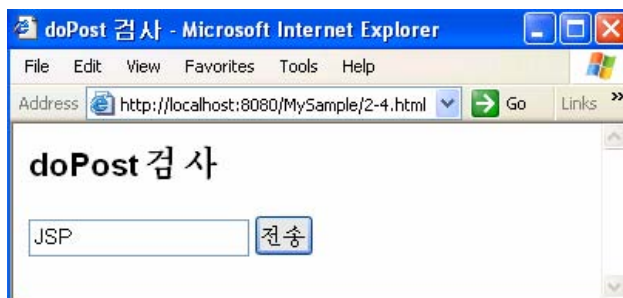



그림 2-13. 실례 2-4의 실행결과

콤파일이 완료되면 아래의 주소로 Servlet에 접근할수 있다.(그림 2-13)
http://localhost:8080/MySample/HelloDoPost.html

프로그람설명

작성한 HTML문서를 실행시키면 우의 그림과 같이 하나의 본문마당과 《전송》단추가 있는 홈이 만들어진다. 본문마당에 글자를 입력한 후 《전송》단추를 찰칵하면 HTML파일의 홈에서 설정한 주소로 Post방식의 의뢰기요청이 이루어진다.

우의 웨브열람기에서 그림과 같이 본문마당에 《JSP》라고 입력한 후 《전송》단추를 찰칵하면 HTML문서의 Form끄리표에서 action속성으로 설정한 《/MySample/servlet/HelloDoPost》로 값을 넘기며 이때 HelloDoPostServlet의 doPost메쏘드는 자동적으로 호출된다. 이때 request와 response는 자동적으로 Servlet의 참조파라메터로 넘어온다. service메쏘드는 의뢰기의 요청방식을 확인하고 doPost메쏘드를 자동호출하게 된다.

doPost메쏘드의 형식은 다음과 같다.

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

다음 과정은 doGet메쏘드와 동일하다. 즉 doPost메쏘드의 참조파라메터 HttpServletRequest와 HttpServletResponse는 자동적으로 넘어오게 되며 역시 이 두 참조파라메터는 의뢰기로부터의 요청과 응답을 클라스로 넘긴다. 실례에서는 Post방식으로 의뢰기가 자료를 넘길 때 이것을 참조파라메터인 request로부터 얻어내고있다.

```
String name = request.getParameter("name");
```

그리고 응답을 의뢰기에게 보내는데 리용하는 흐름을 얻기 위하여 response객체로부터 다음과 같은 메쏘드를 사용한다.

```
PrintWriter out = response.getWriter();
```

제3장. JSP의 기초

이 장에서는 JSP의 기본적인 내용 즉 JSP의 동작과 주기 그리고 JSP사용방법을 학습한다.

제1절. JSP의 개요

JSP는 HTML로만 이루어진 정적인 웨브페지의 결함을 퇴치하여 동적인 웨브페지를 보다 효률적으로 만들수 있게 하는 봉사기측스크립트언어이다. JSP내부에서 Java를 그대로 사용할수 있으며 스크립트방식으로 프로그람을 작성할수 있으므로 편리하고 효과적이다. JSP의 목적은 동적인 웨브페지를 효률적으로 만들고 응용하기 위한 방법을 제공하는 것이다. 동적인 웨브페지와 정적인 페지의 특징은 다음과 같다.

● **정적인 페지의 특징**

HTML은 정적인 문서이다. 정적인 웨브페지는 의뢰기에서 봉사기측에 문서를 요구할 때 이미 작성된 문서를 의뢰기에 그대로 봉사한다.

● **동적인 페지의 특징**

원하는 정보를 의뢰기가 요청할 때 봉사기에서 실시간으로 작업을 처리하여 보다 동적으로 의뢰기에게 봉사한다. 봉사기에서는 자료기지작업이나 의뢰기가 원하는 작업을 처리하고 그 결과만을 의뢰기에 보낸다.

봉사기에서 작업을 처리한다는 의미에서 Server Page라는 말을 쓰고있으며 Java를 리용하고있으므로 Java Server Page라고 하는것이다. 동적인 페지와 정적인 페지의 동작과정에 대하여 그림 3-1에서 보여주었다.

동적페지의 봉사기에서는 의뢰기가 요구하는 요청을 처리하는데 이때 리용되는 프로그람이 바로 JSP이다.

왜 JSP가 동적인 웨브페지를 만드는데 아주 효률적인가?

① JSP는 Java를 봉사기환경에서 사용하는 스크립트방식의 언어이다.

② JSP는 사용하는 언어가 Java로 되여있으므로 어떤 가동기반이나 어떤 웨브봉사기에서도 사용가능하다.

③ JSP는 단일토막으로 의뢰기의 요청에 봉사한다. 요청이 있을 때마다 프로쎄스를 생성하는 기존의 CGI와는 달리 하나의 기억기를 공유하면서 봉사하므로 봉사기측의 부하를 줄여준다.

④ JSP내부에는 보여주는 코드만 작성하고 직접 작업하는 부분은 JavaBean으로 구성하여 분리할수 있다. 이렇게 함으로써 서로 영향을 주지 않고 수정할수 있는 우점을 가지고있다. 또한 Java의 우점인 재리용성을 높일수 있다.

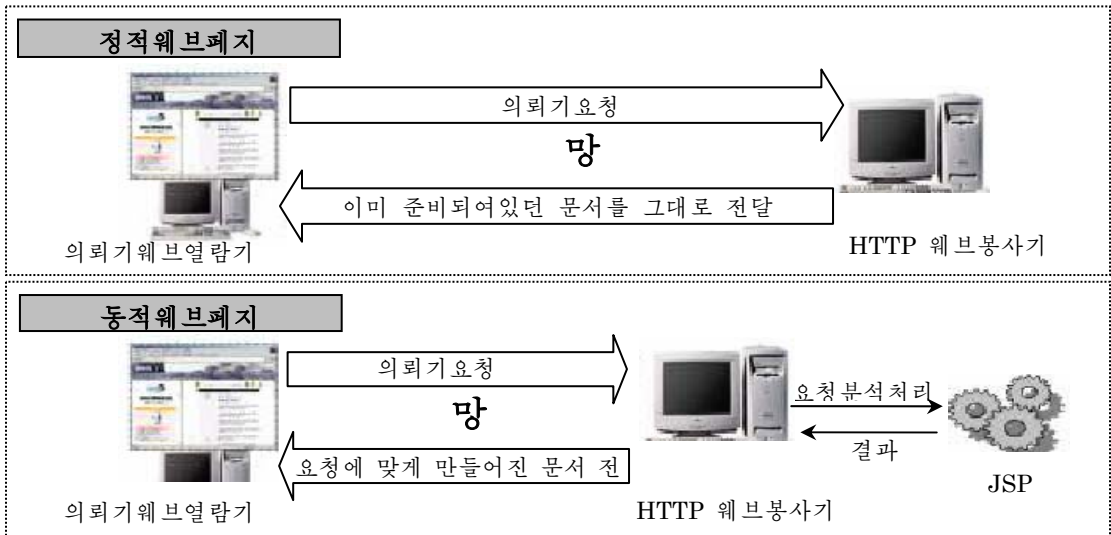



그림 3-1. 정적 및 동적웨브페지에서 봉사과정

제2절. JSP의 동작과정

JSP를 실행시키기 위해서는 JSP스크립트가 포함된 JSP파일을 Servlet파일로 변환하는 과정을 거쳐야 한다. 이 작업은 의뢰기의 요청이 있을 때 또는 JSP파일이 변경되였을 때 새로운것을 적재하기 위해서 진행된다. 한번 적재되면 그 다음부터는 적재된 Servlet객체를 재리용한다.

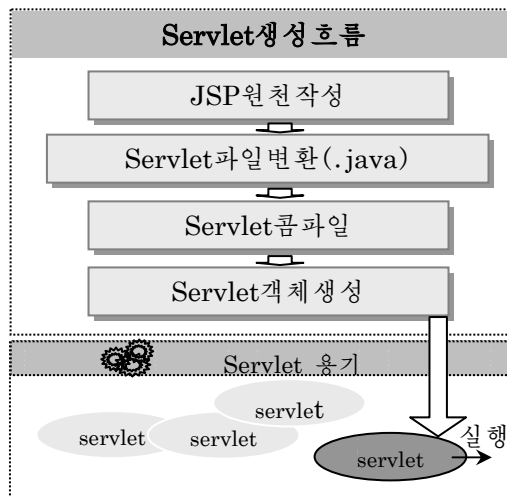



그림 3-2. Servlet의 생성흐름

JSP가 Servlet파일로 변환되면 콤파일과정을 거친다. 즉 Servlet파일은 .java형태로 되며 Java파일을 실시간적으로 콤파일하게 된다. 다음 Servlet용기는 해당 클라스에 대한 객체를 생성하여 적재한다. 적재된 객체는 봉사를 시작한다. 이와 같은 과정을 그림으로 나타내면 아래와 같다.(그림 3-2)

제3절. JSP내부의 생명주기

jsp변환 Servlet프로그람은 jspInit(), _jspService(), jspDestroy()인 3가지 메쏘드에 의하여 초기화, 봉사, 파괴의 과정을 거친다. 여기서 jspInit메쏘드는 한번만 호출되고 봉사요청이 있을 때마다 _jspService메쏘드가 호출되게 된다.(그림 3-3)

의뢰기에서 jsp파일을 찰각하면 jsp는 Servlet로 변환되며 그 변환된 Servlet에서jspInit메쏘드가 제일 먼저 호출된다. jspInit메쏘드는 JSP에서 변환된 Servlet의 초기화 및 봉사를 시작하기 위한 준비를 하는 메쏘드이다. 그리고 Servlet에서 init메쏘드가 service메쏘드를 호출하듯이 JSP의 jspInit메쏘드는 _jspService메쏘드를 호출한다. 호출된 _jspService메쏘드는 의뢰기에서 요구한 작업을 수행한다.

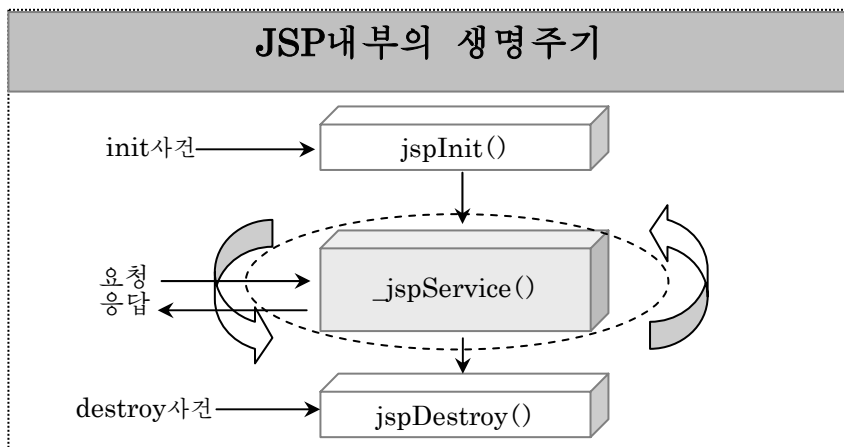



그림 3-3. JSP의 생명주기

두번째 의뢰기의 요청이 있을 때부터는 _jspService메쏘드를 호출하여 의뢰기의 요청에 응답하게 된다. _jspService메쏘드는 의뢰기의 요청방식인 Get와 Post방식을 모두 처리할수 있다.

Servlet객체가 더 이상 봉사를 할 필요가 없는 경우 기억기에서 제거되며 이때 호출되는 메쏘드는 jspDestroy메쏘드이다. jspDestroy메쏘드가 호출되면 페품회수기는 객체의 기억기를 제거한다. jspInit와 jspDestroy는 한번만 호출되며 _jspService메쏘드는 요청이 있을 때마다 호출된다.

제4절. JSP에서 생성되는 Java파일

JSP가 실행되면 Java(.java)와 클라스(.class)로 변환되며 그에 따라 페지를 동적으로 생성한다. 간단한 JSP프로그람을 만들어보자. 이미 hello.jsp를 만들어 보았지만 여기서는 다시 출발하는 의미에서 0 - 9까지 출력하는 프로그람을 작성해보자.

실례 3-1.

jspproof.jsp

```
<%@page contentType="text/html;charset=big5" %>
<HTML><HEAD><TITLE>JSP Proof</TITLE></HEAD><BODY>
<H1> JSP Proof</H1>
<%
  for(int i=0;i<10;i++){
    out.println("<font color=blue>JSP Proof</font>: "+i);
    out.println("<BR>");
  }
%>
</BODY></HTML>
```

```
C:\Tomcat\webapps\MySample>dir jspproof.jsp
2007-11-2   12:51a              224 jspproof.jsp
```

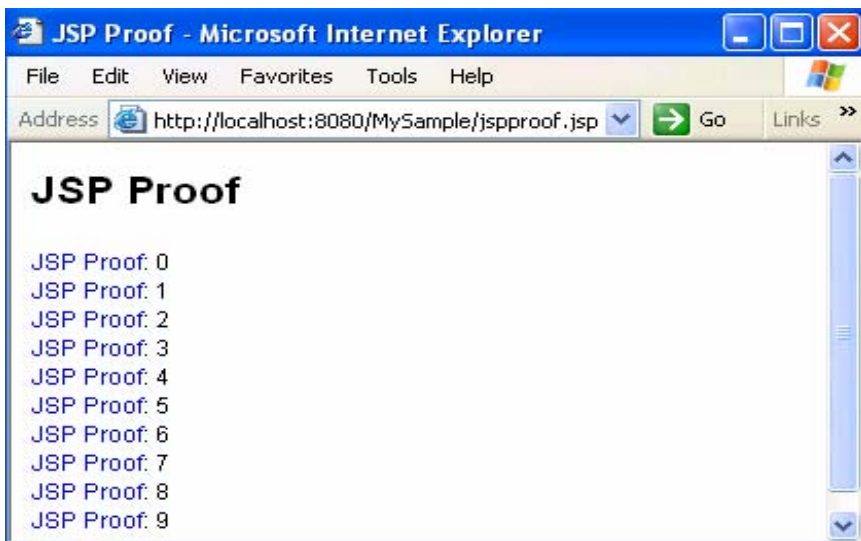



그림 3-4. 실례3-1의 실행결과

우의 JSP파일에 접근하려면 다음과 같은 주소로 접근하면 된다.

http://localhost:8080/MySample/jspproof.jsp

프로그람설명

조선글을 사용하기 위하여 다음과 같은 코드를 삽입하였다.

<%@page contentType="text/html;charset=big5" %>

JSP가 java와 class파일로 변환되였는가를 알려면 work등록부를 살펴보아야 한다. 즉 Tomcat/work/MySample등록부안에 .java와 .class의 확장자를 가진 파일이 있는가를 본다.(그림 3-5)

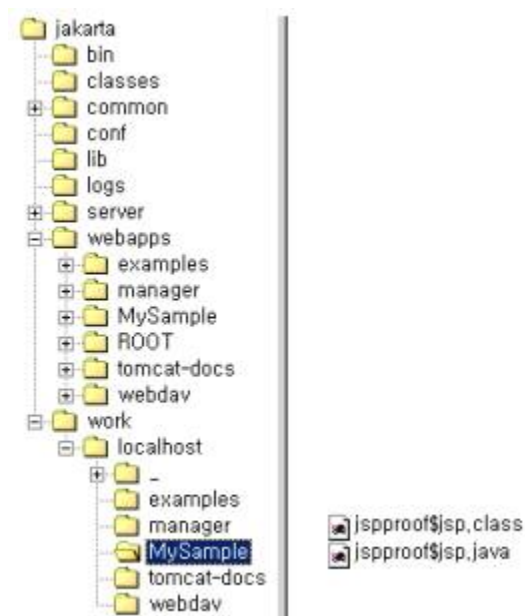



그림 3-5. JSP의 변환파일등록부위치

work등록부는 림시등록부의 역할을 하므로 .jsp가 적재될 때 .java로 변환되고 .class로 변환한 파일이 그대로 남아있다. Tomcat의 판본마다 파일의 이름이 다를수 있다. 그러나 처음 적재될 때 이 파일들은 만들어진다. 그리고 수정되였을 때 이 작업은 한번 더 진행된다.

JSP를 만들고 실행한 후 문제가 발생하여 오유제거를 하려고할 때 문제점을 찾지 못했을 경우 이 Work등록부를 지우고 봉사기를 다시 실행해볼수 있다. 그 리유는 Tomcat엔진의 상태에 따라서 새로 작성한 jsp가 .class파일로 변환되지 않고 전에 쓰이던것이 계속 리용될수 있기때문이다. work등록부는 림시등록부이므로 지우더라도 Tomcat봉사기를 다시 시동시키면 자동적으로 다시 생성된다.

제5절. JSP에서 생성된 Java파일의 구조

우에서 본것처럼 Tomcat의 림시등록부인 work등록부에는 JSP파일이 Java로 변환된것과 Class파일로 콤파일된것이 함께 들어있다. 그 Java파일을 실례들어 JSP파일이 어떻게 Java파일로 만들어지는가를 보기로 하자. 우선 JSP파일로서 앞에서 사용한 hello.jsp를 리용한다. hello.jsp를 요청했을 때 work등록부에 자동적으로 생성된 hello$jsp.java파일을 간단히 편집기로 열어보자.

실례 3-2

hello.jsp

```
<HTML><HEAD><TITLE>Hello JSP</TITLE></HEAD><BODY>
<H1> Hello JSP Test</H1>
<%
    out.println("<font color=blue>Hello World! JSP</font>");
%>
</BODY></HTML>
```

그림 3-6은 실례 3-2의 실행결과이다

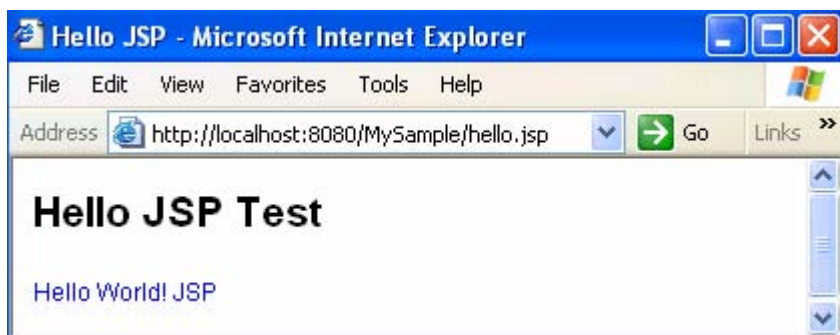



그림 3-6. 실례 3-2의 실행결과

아래의 실례는 우의 jsp파일이 java파일로 변환된것이다.

실례 3-3

hello$jsp.java

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
public class hello$jsp extends HttpjspBase {
 static{}
public hello$jsp( ){}
private static boolean _jspx_inited = false;
public final void _jspx_init()
        throws org.apache.jasper.runtime.jspException{}
public void _jspService(HttpServletRequest request,
        HttpServletResponse  response)
        throws java.io.IOException, ServletException{
jspFactory_jspxFactory = null;
PageContext pageContext = null;
HttpSession session = null;
ServletContext application = null;
ServletConfig config = null;
jspWriter out = null;
Object page = this;
String  _value = null;
try {
    if (_jspx_inited == false) {
        synchronized (this) {
            if (_jspx_inited == false) {
                _jspx_init();
                _jspx_inited = true;
            }
        }
    }
    _jspxFactory = jspFactory.getDefaultFactory();
    response.setContentType("text/html;charset=ISO-8859-1");
    pageContext = _jspxFactory.getPageContext(this, request, response, "", true, 8192, true);
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
```

```
        out = pageContext.getOut();
        // HTML // begin [file="/hello.jsp";from=(0,0);to=(2,0)]
      out.write("<HTML><HEAD><TITLE>Hello JSP</TITLE></HEAD><BODY>\r\n
      <H1> Hello JSP Test</H1>\r\n");
       // end
       // begin [file="/hello.jsp";from=(2,2);to=(4,0)]
       out.println("<font color=blue>Hello World! JSP</font>");
       // end
       // HTML // begin [file="/hello.jsp";from=(4,2);to=(8,0)]
       out.write("\r\n</BODY></HTML>\r\n\r\n\r\n");
       // end
 } catch(Throwable t) {
       if (out != null && out.getBufferSize() != 0)
          out.clearBuffer();
       if (pageContext != null)
          pageContext.handlePageException(t);
} finally {
       if (_jspxFactory != null)
          _jspxFactory.releasePageContext(pageContext);
      }
   }
}
```

```
C:\jakarta\work\localhost\MySample>dir *.java
2007-11-4  01:41a                 2,593 Hello$jsp.java
```

프로그램설명

먼저 필요한 클라스에 대한 몇가지 패키지들을 적재하였다.

import javax.servlet.*;

import javax.servlet.http.*;

import javax.servlet.jsp.*;

import org.apache.jasper.runtime.*;

그리고 hello$jsp클라스는 HttpjspBase를 계승하고있다.

즉 public class hello$jsp extends HttpjspBase

또한 hello$jsp클라스의 구성자부분이 비여있고 _jspx_init메쏘드도 비여있다는것을 알수 있다.

public hello$jsp( ) {}

public final void _jspx_init() throws org.apache.jasper.runtime.jspException {}

가장 많은 부분을 차지하는 _jspService메쏘드를 볼수 있다.

public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException, ServletException {

_jspService메쏘드는 HttpServletRequest와 HttpServletResponse형의 참조파라메터를 가지고있다.

또한 jsp파일안에 직접 작성한 내용이 아래와 같이 변환되여있는것도 볼수 있다.

일반적인 HTML꼬리표들과 <% %>꼬리표안에 있던 모든 요소들이 out객체에 의해 기록되여있다. 물론 out는 _jspService안의 지역변수로 되여있다.

_jspService메쏘드의 일부분

```
// HTML // begin [file="/hello.jsp";from=(0,0);to=(2,0)]
out.write("<HTML><HEAD><TITLE>Hello JSP</TITLE>
</HEAD><BODY>\r\n<H1> Hello JSP Test</H1>\r\n");
// end
// begin [file="/hello.jsp";from=(2,2);to=(4,0)]
out.println("<font color=blue>Hello World! JSP</font>");
// end
// HTML // begin [file="/hello.jsp";from=(4,2);to=(8,0)]
out.write("\r\n</BODY></HTML>\r\n\r\n\r\n");
// end
```

제6절. JSP내부 _jspService메쏘드의 구조

_jspService()메쏘드의 내부를 고찰하자. _jspService()메쏘드는 참조파라메터로 request와 response를 가지고있다. request는 HttpServletRequest, response는 HttpServletResponse형이다. 우선 jsp의 <% %>의 내용과 HTML꼬리표들이 이 메쏘드안에 삽입된다. 그리고 여러개의 지역변수들이 선언되여 사용된다. _jspService의 구성요소를 살펴보면 다음과 같다.

참조파라메터

1. HttpServletRequest request
2. HttpServletResponse response

필요한 지역변수

```
1. PageContext pageContext = null;
2. HttpSession session = null;
3. ServletContext application = null;
4. ServletConfig config = null;
5. jspWriter out = null;
6. Object page = this;
7. jspFactory _jspxFactory = null;
8. String  _value = null;
```

지역변수초기화

```
1. _jspxFactory = jspFactory.getDefaultFactory();
2. pageContext = _jspxFactory.getPageContext(this, request, response, "", true, 8192, true);
3. application = pageContext.getServletContext();
4. config = pageContext.getServletConfig();
5. session = pageContext.getSession();
6. out = pageContext.getOut();
```

여기서 초기화된 변수와 참조파라메터들을 **내장객체**라고 한다.

_jspService메쏘드부분을 직접 보면서 설명하도록 하자. hello.jsp에서 해당되는 _jspService메쏘드부분은 아래와 같다.(실례 3-5)

실례 3-4

hello.jsp
<HTML><HEAD><TITLE>Hello JSP</TITLE></HEAD><BODY> <H1> Hello JSP Test</H1> <%   out.println("Hello World! JSP"); %> </BODY></HTML>

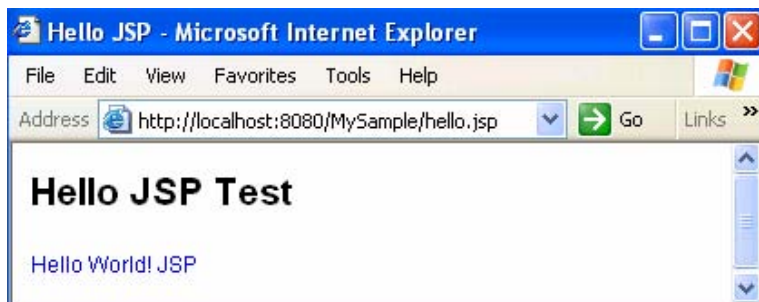



그림 3-7. 실례 3-4의 실행결과

실례 3-5

_jspService메쏘드부분

```
public void _jspService(HttpServletRequest request, HttpServletResponse
  response) throws java.io.IOException, ServletException {
    jspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    jspWriter out = null;
    Object page = this;
    String  _value = null;
    try {
      if (_jspx_inited == false) {
         synchronized (this) {
            if (_jspx_inited == false) {
             _jspx_init();
             _jspx_inited = true;
            }
         }
      }
     _jspxFactory = jspFactory.getDefaultFactory();
     response.setContentType("text/html;charset=ISO-8859-1");
     pageContext = _jspxFactory.getPageContext
                        (this, request, response,  "",true, 8192, true);
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
       // HTML // begin [file="/hello.jsp";from=(0,0);to=(2,0)]
      out.write("<HTML><HEAD><TITLE>Hello
      JSP</TITLE></HEAD><BODY>\r\n<H1> Hello JSP Test</H1>\r\n");
      // end
      // begin [file="/hello.jsp";from=(2,2);to=(4,0)]
      out.println("<font color=blue>Hello World! JSP</font>");
      // end
      // HTML // begin [file="/hello.jsp";from=(4,2);to=(8,0)]
      out.write("\r\n</BODY></HTML>\r\n\r\n\r\n");
```

```
        // end
    } catch (Throwable t) {
      if (out != null && out.getBufferSize() != 0)
          out.clearBuffer();
      if (pageContext != null) pageContext.handlePageException(t);
    } finally {
      if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
    }
}
```

프로그람설명

메쏘드로서 IOException, ServletException이 선언되여있다. 이것은 jsp파일이 Servlet 파일로 자동적으로 바꾸어지는 과정에 례외선언이 되여있다고 볼수 있다. 그리고 메쏘드 내부에서 지역변수를 선언하고있다.

선언된 객체변수를 초기화한 다음 _jspService메쏘드안에서 사용한다.

지역객체변수와 jsp에서 사용되는 끄리표사이에는 일정한 관계가 있다.

- <% %>끄리표와 _jspService의 지역변수

jsp문서의 <% %>끄리표안에 들어있는 부분은 _jspService메쏘드안에 삽입된다.

<% %>안에서 _jspService메쏘드안의 지역변수를 모두 사용할수 있다.

여기에 선언된 pageContext, session, application, config, out, page와 참조파라메터 request, response은 내장객체이다. <% %>끄리표안에서는 _jspService에 존재하는 out객체를 리용하여 의뢰기에 변수를 출력한다.

표 3-1에서는 JSP내장객체들에 대하여 소개한다.

표 3-1. JSP내장객체

JSP내장객체	
ServletRequest request (HttpServletRequest request)	봉사기로 http요청을 전송할 객체
ServletResponse response (HttpServletResponse response)	의뢰기로 응답을 전송할 객체
PageContext pageContext	다른 내장객체를 얻거나 요청을 처리할 조종권을 다른 페지로 위임하는 객체
HttpSession session	의뢰기와 봉사기와의 대화접속자료를 가지고 있는 객체

JSP내장객체	
ServletContext application	Web응용프로그람이 실행되는 실행환경에 대한 정보를 담고있는 객체
jspWriter out	Servlet가 요청을 처리하여 응답을 전송할 때 전송할 응답에 대한 출력흐름 객체
ServletConfig config	Servlet객체가 참조하게 될 초기설정자료를 담고있는 객체
Object page (HttpjspPage)	Servlet객체를 참조하는 참조변수
Throwable exception	례외가 발생하는 경우 오유페지에 전달되는 객체

제7절. JSP에서 성원변수와 성원메쏘드

jsp는 실행되면 .java파일로 변환되고 그 파일은 Tomcat의 work등록부에 있게 된다. jsp파일에 작성한 부분은 전부 _jspService메쏘드안으로만 들어가지 않는다.

jsp가 실제 실행될 때 하나의 클라스로 이루어진다는것을 학습하였다. 그 구조는 jspInit, _jspService, jspDestroy인 3개의 성원메쏘드로 되여있다.

그리고 <% %>안의 모든 구문과 HTML코드들은 전부 _jspService로 들어가지만 클라스의 성원메쏘드와 성원변수 차원에서 클라스안에 삽입할수 있다. 이것을 지원하는것이 바로 <%! %>꼬리표이다. 이것을 선언문이라고 한다.

jsp에서 클라스의 성원변수와 성원메쏘드를 삽입할 때 사용하는 구문

<%! %> → 선언문

이와 같은 구조를 그림으로 나타내면 그림 3-8과 같다.

일반적으로 의뢰기가 요청할 때 _jspService만 호출되지만 클라스에 메쏘드를 삽입하여 처리할수도 있다. JSP는 하나의 객체를 공유하면서 _jspService만을 호출하기 때문에 만약 <%! %>선언문꼬리표를 리용하여 성원변수를 삽입하면 대역변수역할을 하게 된다.

이것을 증명하기 위하여 앞의 hello.jsp실례를 조금 변경시킨 실례를 고찰한다.(실례 3-6)

물론 이 실례에서는 <%! %>선언문꼬리표와 <% %>스크립트레트꼬리표를 둘다 사용한다. 이것들사이의 관계를 보자.

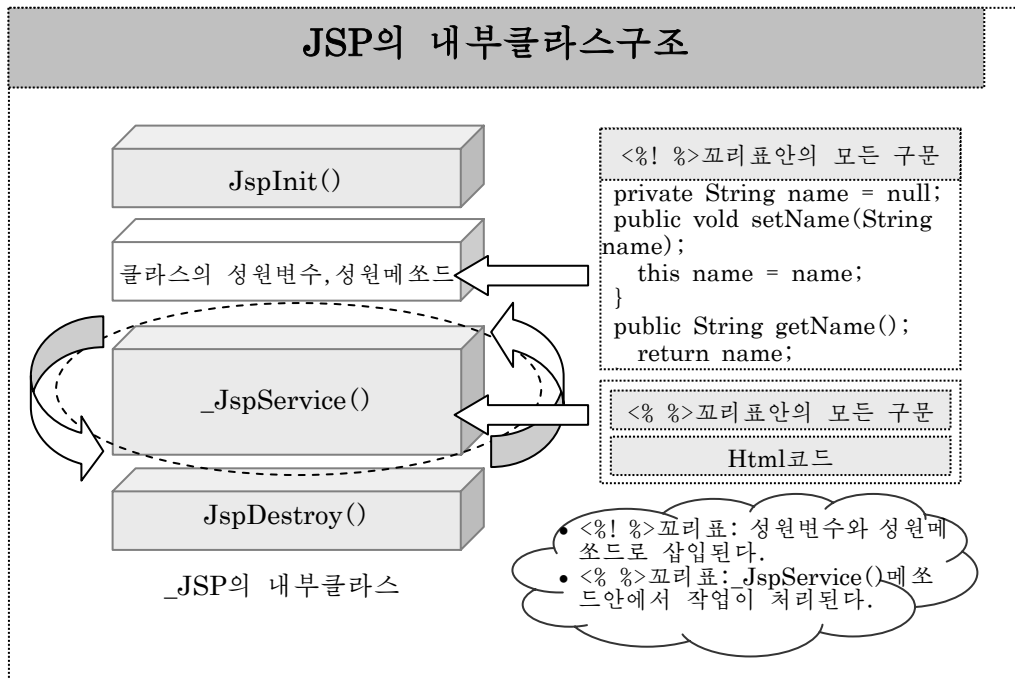



그림 3-8. JSP내부의 클라스구조

실례 3-6

hello1.jsp

```
<HTML><HEAD><TITLE>Hello JSP1</TITLE></HEAD><BODY>
<H1> hello JSP1 Test</H1>
<%!
   private String str = “JSP reference”;
   private static int a = 33;
   public int sum(int b) {
     return b + 3;
   }
%>
<%
    out.println("<font color=blue>" + str + "</font><BR>");
    out.println(sum(a));
%>
</BODY></HTML>
```

앞에서 본 hello.jsp와 달라진 부분이 있다. 바로 <%! %> 구문이 사용된것이다. <%! %> 구문에서 선언한 변수와 메쏘드는 생성된 Java파일(실례 3-7)에서 성원변수와 성원메쏘드의 역할을 하고있으며 _jspService에서는 자신의 성원을 호출하는것처럼 된다.

물론 <% %>안에서 이 성원변수와 성원메쏘드를 사용할수 있다. 그리고 <%! %>선언문안에서는 내장객체를 사용할수 없다는것을 알수 있다.

실례 3-7.

Hello1$jsp.java

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
public class hello1$jsp extends HttpjspBase {
  // begin [file="/Hello1.jsp";from=(2,3);to=(8,0)]
      private  String str = "JSP  reference";
      private  static int  a= 33;
      public  int sum(int b) {
          return b+ 3;
      }
      // end
      static { }
      public hello1$jsp( ) {
      }
      private static boolean _jspx_inited = false;
      public final void _jspx_init() throws org.apache.jasper.runtime.jspException {
      }
      public void _jspService(HttpServletRequest request,
           HttpServletResponse response) throws java.io.IOException, ServletException {
         jspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        jspWriter out = null;
        Object page = this;
        String  _value = null;
        try {
//아래부분 생략
```

아래의 그림 3-9는 우의 실례의 실행결과이다.

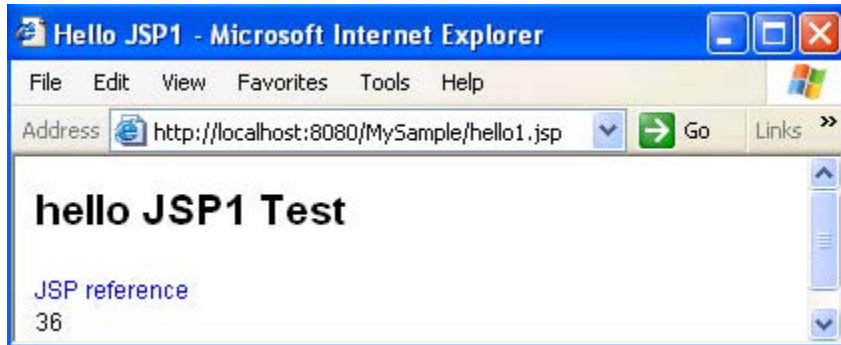



그림 3-9. 실례 3-6의 실행결과

제8절. JSP에서 Bean의 사용

Bean은 일종의 특정한 일을 독립적으로 수행하는 부품이다. 그러나 기존의 클라스들과 다른 어떤 특별한 클라스의 계승이나 대면을 실현한것은 아니다. Bean은 부품이므로 재리용이 가능하다는 우점을 가지고있다. 즉 다른 곳에서 이 부품의 코드를 수정하지 않고 재리용할수 있다.

빈즈를 개발하는데서 다음과 같은 4가지 규칙을 지켜야 한다.

① Bean클라스는 인수가 없는 구성자를 반드시 가져야 한다.

② 성원변수는 공개(public)변수가 되지 말아야 한다.

③ 변수에 대한 접근은 setXxxx()메쏘드나 getXxxx()메쏘드를 통해서만 가능하다.

④ setXxxx()메쏘드, getXxxx()메쏘드는 항상 공개(public)메쏘드이여야 한다.

이 내용들을 자세히 보도록 하자.

첫번째로 Java와 마찬가지로 JavaBean은 인수가 없는 구성자를 통해 초기화된다. 따라서 만들지 않아도 기정구성자를 가지고있기때문에 꼭 쓰지 않아도 되지만 초기화가 필요하다면 반드시 인수없는 구성자를 만들어야 한다.

두번째로 JavaBean에서는 메쏘드를 통해서만 그 내부에 접근할수 있다. 이것은 성원변수를 공개변수로 선언하게 되면 외부에서 이 성원변수를 변경하게 되므로 다른 객체들이 영향을 받게 되는 문제를 막기 위해서이다.

세번째로 접근메쏘드는 getXxxx()메쏘드를, 변경메쏘드는 setXxxx()메쏘드의 형태를 가져야 한다. 그리고 론리형태의 되돌림값을 갖는 메쏘드는 isXxxx()의 형태를 가진다.

네번째로 JavaBean은 메쏘드를 통해서만 접근이 가능하므로 setXxxx()메쏘드, getXxxx()메쏘드는 항상 공개메쏘드이여야 한다. 이것은 JSP용기가 공개메쏘드가 아닌 메쏘드는 인식을 하지 못하기때문이다.

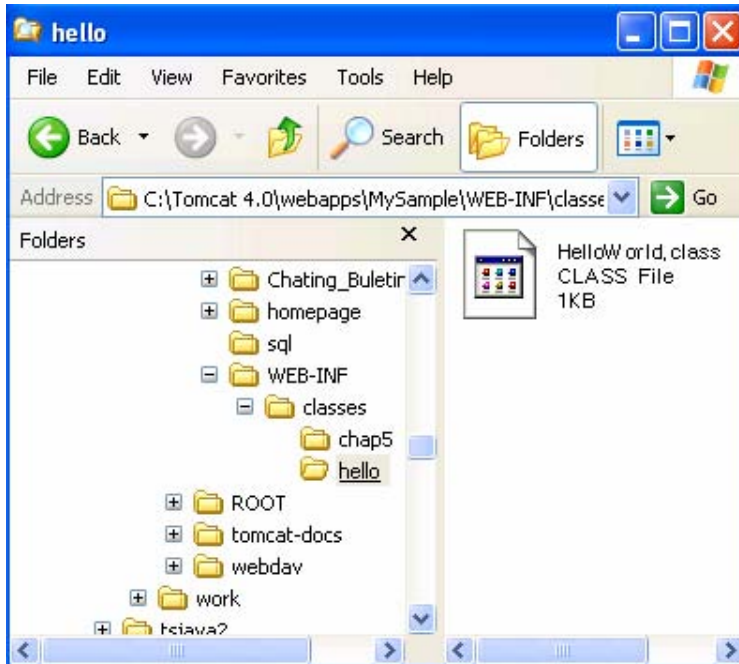



그림 3-10. 패키지의 등록부위치

이제는 Bean의 사용방법에 대해서 보자.

HelloWorld.java를 실례들어 고찰한다. 보통 클라스들을 관리하기 위하여 classes등록부아래에 새로 패키지를 생성한다. 이때 HelloWorld객체가 그림 3-10에서 보여준 등록부에 존재한다면 원천코드에서 처음에 다음과 같이 패키지를 적재해주어야 한다.

<%@page import="hello.*" %>

즉 classes의 아래등록부가 패키지등록부로 된다. import로 패키지를 적재하였으면 객체를 마음대로 생성하여 사용할수 있다. 아래의 실례를 보자.

실례 3-8

HelloWorld.java

```
package hello;
public class HelloWorld {
   private String statement="";
   public HelloWorld()  {
      this.statement = "No Statement";
   }
   public String getStatement()  {
       return statement;
   }
```

```
    public void setStatement(String statement) {
        this.statement = statement;
    }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes>javac -d . HelloWorld.java

프로그람설명

이것은 Bean으로서 사용할 HelloWorld클라스이다. 우에서 설명한 Bean파일작성의 규칙들이 모두 적용되여있다.

① 성원변수를 비공개(private)변수로 설정하였다.

private String statement="";

인수가 없는 구성자를 만들어놓는다. statement변수에 아무런 설정이 없을 경우 "No Statement"를 출력한다.

② 구성자를 만든다면 참조파라메터없는 구성자를 만든다.

public HelloWorld() { }

getXxx(), setXxx()메쏘드의 Xxx에는 성원변수의 이름이 들어간다. 단지 첫번째 X는 대문자로 써주어야 한다.

③ public set, get메쏘드를 만든다.

public String getStatement() { }

public void setStatement(String statement) { }

다음은 HellowWorld클라스를 사용하는 JSP실례이다.

실례 3-9

hellobean2.jsp

```
<HTML>
<%@page import="hello.*" %>
<BODY><BR>
<%
    String str="";
    HelloWorld myhello = new HelloWorld();
    myhello.setStatement("Hello! JSP Bean!!");
    str = myhello.getStatement();
    out.println(str + "<BR>");
    myhello.setStatement("Hello again!!");
    str = myhello.getStatement();
    out.println(str);
%>
<BR></BODY></HTML>
```

C:\Tomcat\webapps\MySample>dir hellobean2.jsp

2007-11-5 07:06p 368 hellobean2.jsp

프로그람설명

먼저 hello패키지안의 클라스를 패키지지시문을 통해 적재한다.

<%@page import="hello.*" %>

자세히 보면 일반적인 Java클라스를 만드는 방법과 별로 차이가 없다. 그러나 jsp에서는 빈을 사용한 객체를 JSP끄리표를 리용하여 생성하는 방법을 많이 쓰고있다.

이 실례의 실행결과는 그림 3-11과 같다.

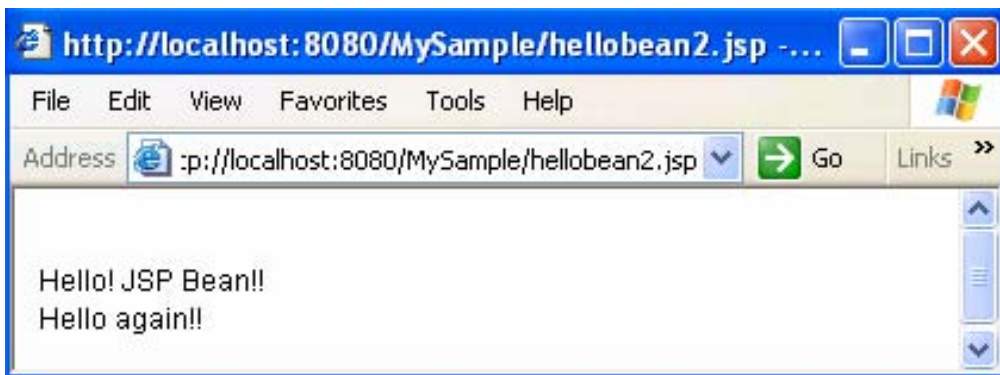



그림 3-11. 실례 3-9의 실행결과

제9절. jspBean끄리표를 리용한 Bean의 사용

JSP에서 객체를 만드는 방법에는 2가지가 있다.

① useBean액션끄리표를 리용하는 방법

② new연산자를 사용하여 객체를 만드는 방법

여기서는 JSP액션끄리표의 하나인 useBean액션끄리표를 통해서 Bean을 적재하는 방법을 고찰한다. useBean끄리표는 새로운 객체를 생성하거나 이미 생성된 객체의 참조를 얻는데 사용된다.

useBean끄리표를 리용하여 HelloWorld라는 Bean클라스의 객체를 생성해보자. 객체의 이름은 myhello이다.

액션끄리표를 리용하는 방법:

```
<jsp:useBean id="myhello" class="HelloWorld" scope="session"/>
```

new연산자를 리용하여 객체를 생성하는 방법:

HelloWorld myhello = new HelloWorld();

이렇게 정의된 myhello라는 Bean객체는 일반적인 객체의 사용과 꼭같은 방법으로 메쏘드를 호출할수도 있고 Bean만이 가지고있는 property메쏘드들을 리용하여 호출할수도 있다.

JSP에서 useBean꼬리표를 사용하여 객체를 생성하고 《Hello World》를 출력하는 실례를 보자.

실례 3-10

HelloBean3.java

```
package hello;
public class HelloBean {
   private String name = "Hello World!!";
   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes>javac -d . HelloBean3.java

실례 3-11

hellobean3.jsp

```
<%@page contentType="text/html;charset=big5" %>
<jsp:useBean id="myHello" class="hello.HelloBean" scope="page"/>
<HTML><HEAD>
<TITLE>HelloBean</TITLE></HEAD>
<BODY >
<center><h3>JSP에서 Bean사용하기</h3></center>
<h2><%=myHello.getName() %></h2>
</BODY>
</HTML>
```

C:\Tomcat\webapps\MySample>dir HelloBean.jsp
2007-11-19 07:06p 368 HelloBean.jsp

그림 3-12. 실례 3-11의 실행결과

프로그람설명

실행된 화면을 보면 new를 통해 객체를 생성한 방법과 별로 차이가 없다. 그러나 JSP 코드를 보면 더 간단해졌다는것을 알수 있다. Java Bean을 리용하기 위하여 useBean끄리표를 사용하고있다. id는 객체실례를 식별할 이름으로 객체를 선언한것과 같다. 여기서는 myHello라는 객체를 생성하고있다. class에는 패키지이름까지 모두 서술한 클라스이름을 넣고 scope에는 Bean이 존재할 범위를 정한다. 여기서는 page로 설정함으로써 Bean을 생성한 페지안에서만 사용할수 있도록 하고있다. scope가 지정되지 않았으면 기정값 page로 설정된다.

<jsp:useBean id="myHello" class="hello.HelloBean" scope="page"/>

선언한 객체 myHello를 통해서 HelloBean클라스의 getName()메쏘드를 호출한다.

<%=myHello.getName() %>

그림 3-12는 이 실례의 실행결과이다.

제10절. JSP Bean끄리표를 리용한 참조파라메터값의 자동설정

우에서 기본적으로 useBean끄리표만을 사용하여 Bean의 값을 리용하는것을 보았다. 이번에는 JSP액션끄리표들을 리용하여 참조파라메터로 넘어오는 값을 Bean끄리표를 리용하여 자동적으로 설정하는 방법을 고찰한다.

Bean을 사용하여 참조파라메터값을 자동적으로 설정하기 위해서는 2개의 끄리표가 더 사용된다. 즉 jsp:setProperty와 jsp:getProperty액션끄리표이다. jsp:setProperty 액션끄리표는 변수를 변경시키는 역할을 하고 jsp:getProperty액션끄리표는 Bean의 변수에 접근하는데 사용한다.

다음의 실례는 Bean을 사용해서 단순히 사용자입력을 받아 출력해주는 프로그람이다. NameBean.html에서 이름을 입력받아 Bean을 리용해서 처리한 다음 JSP화면에 출력하는 실례이다. NameBean.html은 이름을 입력받는 간단한 HTML문서이다.

실례 3-12

NameBeam.java

```
package jsp;
public class NameBean{
    private String name;
    public String getName(){
      return name;
    }
    public void setName(String name){
      this.name = name;
    }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes>javac -d . NameBean.java

NameBean클라스는 JSP에서 사용할 Bean클라스이다. 이 클라스에는 setName()과 getName()메쏘드가 있다.

실례 3-13

NameBean.html

```
<HTML>
<HEAD><TITLE> NameBean</TITLE></HEAD>
<BODY>
<Form action="name.jsp">
이름  : <input type="text" Name="name">
<input type="submit" value="전송">
<input type="submit" value="취소">
</BODY>
</HTML>
```

이 실례의 실행결과는 그림 3-13과 같다.

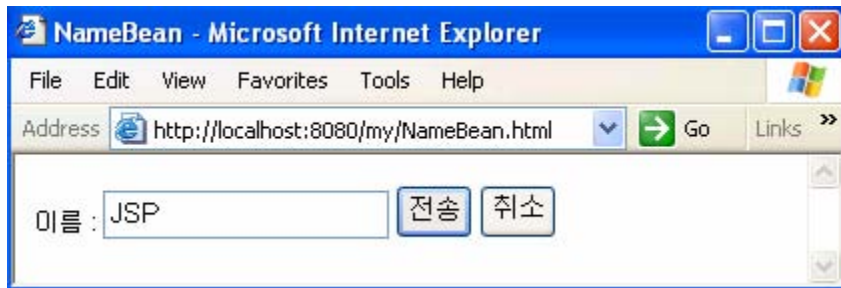



그림 3-13. 실례 3-13의 실행화면

실례 3-14

Name.jsp

```
<%@ page contentType="text/html;charset=big5"%>
<jsp:useBean id="nameTest" class="jsp.NameBean" />
<HTML>
<HEAD><TITLE> NameBean Test </TITLE></HEAD>
<BODY>
<CENTER><H1>이름  Test</H1></center>
<jsp:setProperty name="nameTest" property="name"  /> <BR>
<hr>이름 ：
<jsp:getProperty name="nameTest" property="name" /> <BR>
<hr>
</BODY></HTML>
```

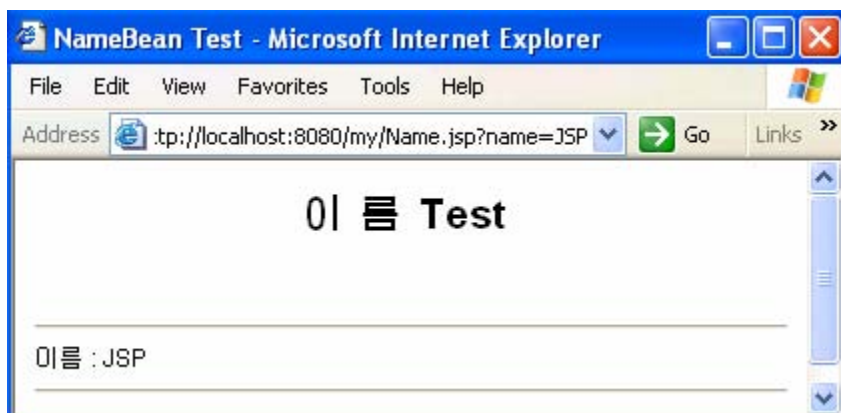



그림 3-14. 실례 3-14의 실행결과

프로그람설명

Name.jsp파일은 사용자입력을 처리하는 JSP페지이다. 먼저 jsp:useBean액션끄리표를 리용하여 NameBean클라스의 객체를 만든다. setProperty와 getProperty를 사용하기 위하여 useBean을 먼저 서술한다.

<jsp:useBean id="nameTest" class="jsp.NameBean" />

setProperty액션끄리표를 리용하여 Bean의 값을 설정한다. 아래의 구문은 Java언어에서 “nameTest.setName();” 과 같은 의미이다.

<jsp:setProperty name="nameTest" property="name" />

여기서 name에는 useBean의 id에 대입한 값을 입력해야 하고 property에는 Bean클라스의 setXxx메쏘드에 대응하기 위해 Xxx에 해당하는 이름을 넣는다.

getProperty액션끄리표를 리용하여 Bean의 값을 출력한다. 아래의 구문은 Java언어에서 “nameTest.getName();” 과 같은 의미이다.

<jsp:getProperty name="nameTest" property="name" />

여기서 name에는 useBean의 id에 대입한 값을 입력하며 property에는 우에서와 마찬가지로 Bean클라스의 getXxx메쏘드에 대응하기 위해 Xxx에 해당하는 이름을 넣는다.

그림 3-15는 HTML과 JSP, JavaBean사이의 관계를 표현한것이다.

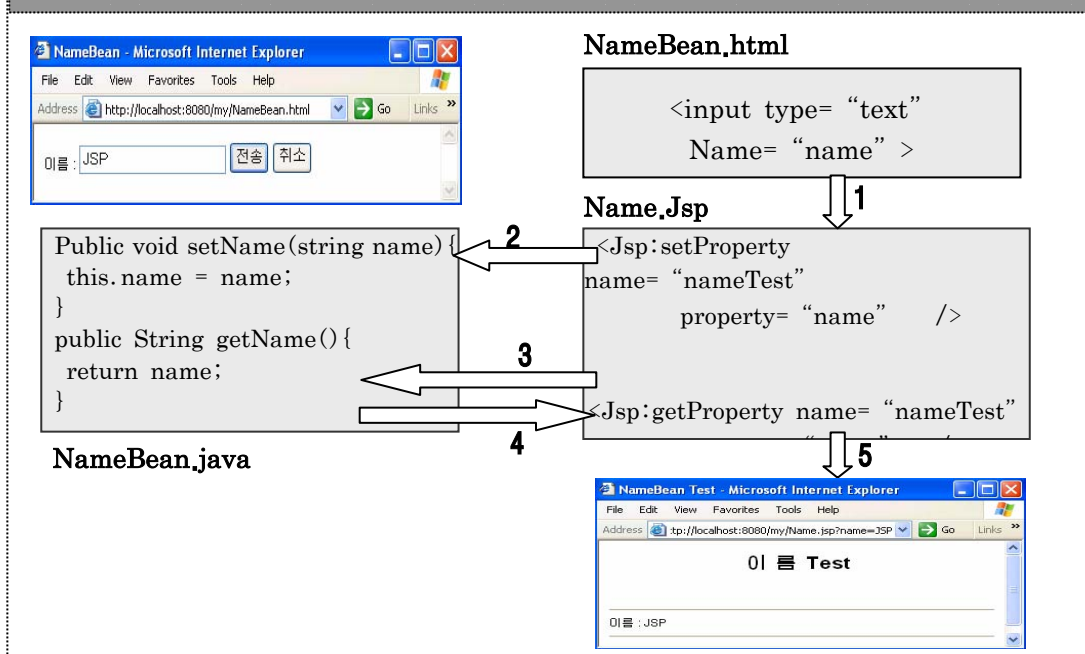



그림 3-15. 액션끄리표를 리용하여 참조파라메터값을 자동적으로 처리하는 순서

NameBean.html에서 홈의 name에 입력한 자료는 Name.jsp에 전송되고 Name.jsp에서는 전달받은 자료를 jsp:setProperty를 리용하여 NameBean클라스에 저장한다. 그리고 저장된 자료를 jsp:getProperty를 리용하여 출력한다.

제11절. JSP에서 Get방식과 Post방식의 처리

웨브열람기와 봉사기사이에 정보를 주고받을 때 사용하는 방식으로 Get방식과 Post방식이 있다.

Get방식은 열람기의 URL주소칸에 파라메터에 대한 값을 입력하여 자료를 전송하는 방식이다. 이 방식은 2가지 결함이 있다. URL주소칸에 보내는 자료가 현시되므로 보안이 되여있지 않다. 둘째로 보낼수 있는 자료의 크기가 1024byte로 제한되여있어 그 이상의 자료는 Get방식으로 보낼수 없다.

Post방식은 Get방식과는 달리 자료의 크기에 제한없이 보낼수 있으므로 많은 정보를 처리할수 있다. 또한 URL주소칸에 보내는 정보가 보이지 않으므로 보안성이 좋다. 그러나 요청하는 쪽에서 HTML의 Form꼬리표와 input꼬리표를 반드시 사용하여야 한다.

아래는 Get방식과 Post방식을 동시에 리용한 실례이다. 그림 3-16은 이 실례의 실행결과이다.

실례 3-15

setPhone.jsp

```
<HTML>
<HEAD>
<TITLE>JSP_get_Post_Test</TITLE>
</HEAD>
<BODY>
<FORM action="getPhone.jsp?name=JSP" method="Post">
  <input type="text" name="phone1" size="3">-
  <input type="text" name="phone2" size="4">-
  <input type="text" name="phone3" size="4">
  <BR>
  <input type="submit">
  <input type="reset">
</FORM>
</BODY>
</HTML>
```

그림 3-16. 실레 3-15의 실행화면

프로그램설명

먼저 Post방식을 사용한 부분에 대하여 보기로 하자. Post방식을 사용하려면 HTML에서 FORM끄리표의 method속성에 Post라고 지정하면 된다.

FORM끄리표의 action속성에는 정보를 처리할 URL를 지정한다. Input끄리표에서는 보낼 정보가 들어있는 변수역할을 하는 name속성을 지정해주고 그 값은 사용자로부터 할당받는다.

<FORM action="getPhone.jsp?name=JSP" method="Post">

다음은 Get방식을 사용한 부분에 대하여 보기로 하자. 만일 정보를 URL로 보내고 싶다면 method속성에 Post대신 Get를 지정해주면 된다. 이때 URL과 보내는 정보를 구분하는 구분자로서 《?》를 사용한다. 여기서 name은 파라메터이고 JSP는 보낼 값이다. 만일 추가로 계속 정보를 보내고 싶다면 &기호를 사용하여 파라메터와 값을 입력하면 된다. 그러나 1024byte의 크기를 넘지 말아야 한다.

다음은 전달받은 정보를 처리하는 프로그람이다. 실행결과는 그림 3-17에서 보여주었다.

실례 3-16

getPhone.jsp

```
<%
  StringBuffer phoneNumber=new StringBuffer();
  phoneNumber.append(request.getParameter("phone1"));
  phoneNumber.append("-");
  phoneNumber.append(request.getParameter("phone2"));
  phoneNumber.append("-");
  phoneNumber.append(request.getParameter("phone3"));
  String name=request.getParameter("name");
%>
<HTML>
<HEAD><TITLE>JSP getPost Result</TITLE>
```

```
</HEAD>
<BODY>
<ul>
  <li>name : <%= name %></li>
  <li>phone : <%= phoneNumber.toString() %></li>
</ul>
</BODY>
</HTML>
```

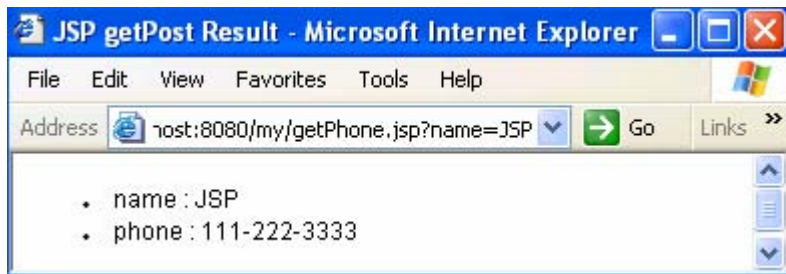



그림 3-17. 실례 3-16의 실행결과

프로그람설명

JSP에서 정보를 받을 때에는 request.getParameter()메쏘드를 사용한다. 메쏘드의 참조파라메터는 setPhone.jsp에서 Post방식의 name속성값이거나 Get방식에서의 파라메터값에 해당된다.

request.getParameter("phone1")

phoneNumber라는 StringBuffer형의 변수에는 Post방식으로 전달된 phone1, phone2, phone3의 값이 저장된다. name이라는 String형의 변수에는 Get방식으로 전달된 name 파라메터의 값이 저장된다. 이렇게 JSP에서는 Servlet에서 보다 간단하게 몇개의 행으로 Post와 Get방식을 처리할수 있다.

제4장. Servlet의 기본

2장에서는 Servlet의 기초적인 내용에 대하여 보았다면 이 장에서는 Servlet에 대하여 구체적으로 설명한다.

제1절. Servlet의 작업과정

4.1.1. 작업과정

Servlet는 주로 HttpServletRequest와 HttpServletResponse로 이루어진다. 이 클라스들은 service메쏘드와 doGet, doPost의 참조파라메터로 넘어오는 인수의 자료형이다. Servlet용기는 의뢰기의 요청이 있을 때 service메쏘드를 호출한다. 이때 의뢰기의 요청과 응답에 해당하는 객체를 참조파라메터형식으로 넘겨주게 되는데 이것이 바로 HttpServletRequest의 request와 HttpServletResponse의 response이다.(그림 4-1)

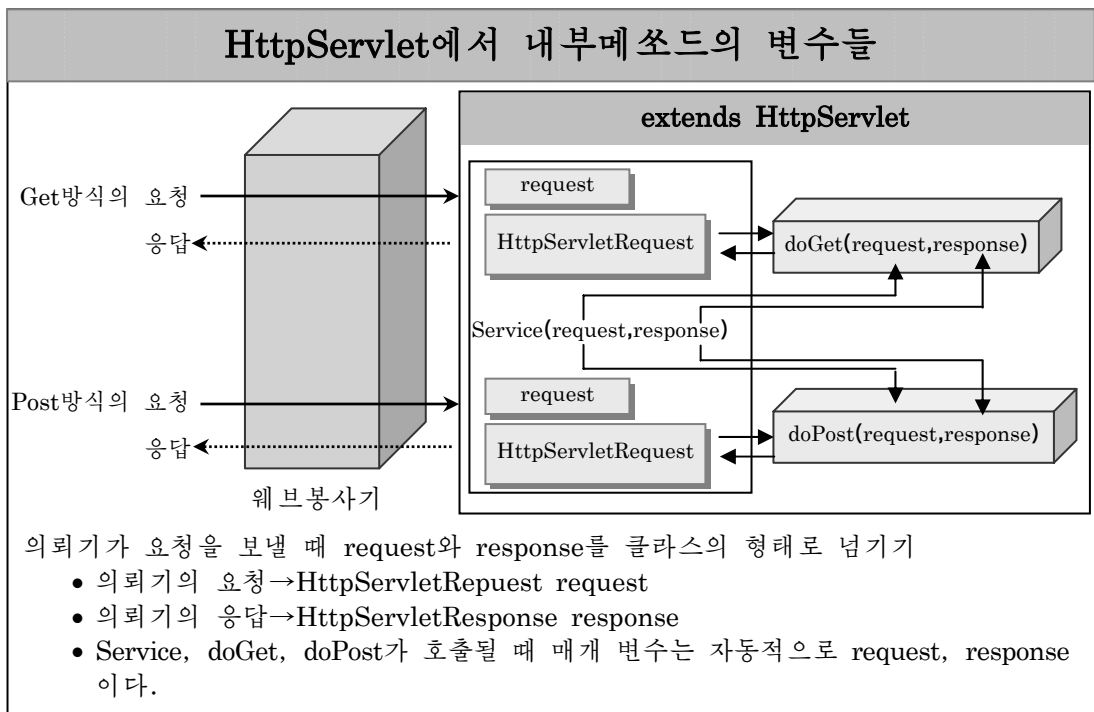



그림 4-1. HttpServlet에서 내부메쏘드의 참조파라메터

Servlet에서 호출되는 service, doGet, doPost는 서로 유기적인 관계를 가지고있다. service는 의뢰기의 요청방식에 따라 doGet과 doPost를 호출한다. service가 호출될 때

HttpServletRequest와 HttpServletResponse는 참조파라메터로 넘어오게 된다. 참조파라메터로 넘어오는 인수를 doGet와 doPost의 참조파라메터인수에 넣어서 재호출하게 된다. 결국 service, doGet, doPost의 참조파라메터는 모두 같다.

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
```

HttpServletRequest는 의뢰기의 요청을 처리하기 위하여 클라스로 넘기기한것이며 HttpServletResponse는 의뢰기에게 응답을 보내기 위하여 클라스형태로 넘기기한것이다.

4.1.2. HttpServletRequest와 HttpServletResponse

의뢰기의 요청을 받아 Servlet용기가 HttpServletRequest객체를 만들고 이 객체를 service메쏘드에 전달한다. service메쏘드는 doGet와 doPost를 호출하여 참조파라메터 형식으로 넘겨주게 된다.

HttpServletRequest는 ServletRequest를 계승하고 HttpServletReponse는 ServletResponse를 계승한다.

ServletRequest와 SevletResponse를 Http규약에 맞게 변형한것이 바로 HttpServletRequest와 HttpServletResponse이다. 이 두 대면이 하는 작업은 다음과 같다.

- HttpServletRequest

① 의뢰기의 모든 요청정보보유

② 머리부정보요청

③ 자료와 질문 파라메터의 형식화

④ InputStream에 의한 의뢰기로부터 전송되는 자료의 접수

⑤ 기타 의뢰기 정보얻기 → 대화접속정보, 쿠키, path ...

- HttpServletResponse

① 의뢰기에게 보내는 모든 정보보유

② 머리부정보응답

- OutputStream(의뢰기에게 보내지는 자료)

① 쿠키설정

② 대화접속설정

제2절. ServletRequest와 HttpServletRequest

ServletRequest는 기본적으로 의뢰기요청에 관한 모든 정보를 가지고있다. 이 대면은 HttpServletRequest로 확장되며 HTTP규약상에서 할수 있는 일들을 포함하고있다. 이 HttpServletRequest는 Servlet의 service에 대한 참조파라메터의 하나로서 Servlet프로그람작성자가 의뢰기의 요청에 관한 작업들을 처리할수 있도록 한다. ServletRequest대면은 다음과 같은 작업을 할수 있다.

– ServletRequest가 하는 일
- 의뢰기자체에 대한 정보추출
- 의뢰기가 전송한 정보추출
- 이러한 구조에 따라서 ServletRequest의 메쏘드를 분류해보면 다음과 같다.

표 4-1. ServletRequest의 메쏘드들(1)

ServletRequest의 성원메쏘드	의뢰기자체에 대한 정보
Object getAttribute(String name)	주어진 이름의 속성값을 얻는다.
String getRemoteAddr()	요청한 의뢰기의 IP주소를 얻는다.
Enumeration getAttributeNames()	이 요청이 가지는 속성들의 이름에 대한 Enumeration형객체를 얻는다.
String getRemoteHost()	의뢰기가 요청한 호스트콤퓨터이름을 얻는다.
void setAttribute(String key, Object o)	주어진 이름의 속성을 설정한다.
String getScheme()	http, https, 또는 ftp 등에서 요청을 위해 사용되는 메쏘드의 이름을 얻는다.
void remvoeAttribute(String key)	주어진 이름의 속성을 제거한다.
String getServerName()	요청을 받은 봉사기의 이름을 얻는다.
String getProtocol()	"HTTP/1.1" 과 같은 형식으로 규약 및 판본을 얻는다.
int getServerPort()	요청을 받은 봉사기의 포구번호를 얻는다.

표 4-2. ServletRequest의 메쏘드들(2)

ServletRequest의 성원메쏘드	의뢰기가 전송한 정보
String getCharacterEncoding()	이 요청에 사용된 문자부호화를 얻는다.
int getContentLength()	이 요청에 포함되여있는 자료의 길이를 구하며 만약 길이를 알수 없는 경우에는 ?1이 귀환된다
String getContentType()	요청내용에 대한 MIME자료형을 얻는다. 이 자료형이 없는 경우에는 null값을 얻는다.

ServletRequest의 성원메쏘드	의뢰기가 전송한 정보
Enumeration getParameterNames()	참조파라메터들의 이름에 대한 Enumeration형객체를 얻는다
String getParameter(String name)	주어진 이름의 참조파라메터가 가지는 값을 얻는다
BufferedReader getReader()	본문을 읽어들이기 위한 BufferedReader객체를 얻는다.
ServletInputStream getInputStream()	요청본체로부터 2진자료를 읽어들이기 위해 한번에 한 행씩 읽을수 있는ServletInputStream객체를 얻는다.

이와 같은 메쏘드들은 HTTP규약에 맞추어져 있는것이 아니라 일반적인 망통신기반에 의하여 사용되는 메쏘드들이다. 그러므로 HTTP규약에 부합되는 대화접속과 쿠키와 같은 정보를 추출하는 작업은 할수 없다. 이러한 리유로 하여 HTTP규약에서 사용되는 대부분의 기능을 가진 HttpServletRequest를 리용한다. 이것을 HTTP규약상에서의 기능별로 분류해보면 다음과 같다.

- **HttpServletRequest의 기능별 분류**
  - request객체의 요청파라메터
  - request객체의 HTTP머리부
  - request객체의 대화접속자료
  - request객체의 쿠키
  - request객체의 요청에 사용된 URL/URI

이러한 분류에 따라 메쏘드를 나누어보면 다음과 같다.(표 4-3~4-7)

표 4-3. request객체의 요청파라메터

request객체의 요청파라메터	
public String getParameter(String name) :	주어진 이름의 참조파라메터가 가지는 값을 얻는다. 지정된 이름의 파라메터가 존재하지 않는 경우 null을 귀환한다.
public Enumeration getParameterNames():	참조파라메터들의 이름에 대한 Enumeration형객체를 얻는다.
public String[] getParameterValues (String name) :	주어진 이름의 참조파라메터가 가지는 모든 값을 문자배렬로 얻는다. 참조파라메터가 다중선택이 가능한 목록(list) 또는 선택칸(choicebox)의 값이라면 여러 개의 값이 하나의 이름으로 전달될수 있지만 참조파라메터가 하나의 값을 가지는 경우라면 getParameter(String name)를 사용하면 된다.

표 4-4. request객체의 HTTP머리부

request객체의 HTTP머리부	
public String getHEADer (String HEADerName)	HTTP요청머리부에 지정된 HEADerName의 값을 문자렬로 귀환한다. 만약 HTTP요청머리부에 HEADerName의 값이 존재하지 않는다면 null을 귀환한다.
Public Enumeration get HEADerNames()	HTTP요청머리부에 포함된 모든 머리부의 이름을 Enumeration형으로 귀환한다.
Public Enumeration get HEADers(String HEADer Name)	HTTP요청머리부에 포함된 HEADerName의 모든 값을 Enumeration형으로 귀환한다.
Public int getIntHEADer (String HEADerName)	HTTP요청머리부에 포함된 HEADerName의 값을 int형으로 귀환한다. 지정된 HEADerName의 값을 int형으로 변환할수 없는 경우 NumberFormat례외가 발생하고 HEADerName머리부가 HTTP요청머리부에 존재하지 않을 경우에는 1을 귀환한다.
public long getDate HEADer(String HEADer Name)	HTTP요청머리부에 포함된 HEADerName의 값을 long형으로 귀환한다. 지정된 HEADerName의 값을 long형으로 변환할수 없는 경우 IllegalArgument례외가 발생하고 HEADerName머리부가 HTTP요청머리부에 존재하지 않을 경우에는 -1을 귀환한다.

표 4-5. request객체의 대화접속자료

request객체의 대화접속자료	
public HttpSession getSession()	요청을 보낸 의뢰기의 HttpSession객체를 얻는다. 이전에 생성된 HttpSession객체가 없었다면 새로운 대화접속객체를 생성한다.
public HttpSession getSession(boolean create)	요청을 시도한 의뢰기의 HttpSession객체를 얻는다. 여기서 create값이 무엇인가에 따라 HttpSession객체를 생성할수도 있고 생성하지 않을수도 있다. 실례로 create가 false로 지정된 경우 해당 의뢰기에 대하여 생성된 HttpSession객체가 없는 경우 null을 귀환한다. create가 true로 지정된 경우 이미 생성된 HttpSession객체를 귀환하고 만일 해당 의뢰기에 생성된 HttpSession객체가 없는 경우 새로운 대화접속객체를 생성하여 귀환한다.

request객체의 대화접속자료	
public String getRequestedSessionId()	요청을 시도한 의뢰기의 대화접속 id를 문자렬로 귀환한다.
public boolean isRequestedSessionId()	요청을 시도한 의뢰기의 대화접속 id가 유효하면 true, 그렇지 않으면 false를 귀환한다.
isRequestedSessionIdFromCookie()	요청을 시도한 의뢰기의 대화접속 id가 쿠키로부터 전달된 경우인가 아닌가에 따라 true 아니면 false를 귀환한다.
isRequestedSessionIdFromURL()	요청을 시도한 의뢰기의 대화접속 id가 URL에 포함된 경우 true 그렇지 않으면 false를 귀환한다.

표 4-6. request객체의 쿠키

request객체의 쿠키
public Cookie[] getCookies() : 의뢰기의 요청에 포함된 쿠키를 쿠키배렬로 귀환한다.

표 4-7. request객체의 요청에 사용된 URL/URI

request객체의 요청에 사용된 URL/URI	
public String getRequestURI()	요청에 사용된 URL에서 URI부분을 문자렬로 귀환한다.
public String getQueryString()	요청에 사용된 질문문자렬을 문자렬로 귀환한다.
public String getMethod()	요청에 사용된 요청방식을 문자렬로 귀환한다.

제3절. SevletResponse와 HttpServletResponse

의뢰기에 응답을 보내기 위한 기초적인 대면은 ServletResponse가 제공한다. 이 대면은 의뢰기로 정보를 보내기 위한 흐름, 완충기크기, Content자료형설정 등의 작업을 할수 있으며 일반적인 망통신에 필요한 메쏘드를 포함하고있다(표 4-8).

표 4-8. ServletResponse의 주요 메쏘드들

ServletResponse의 주요 성원메쏘드	
void flushBuffer()	완충기에 있는 내용을 의뢰기에 전달한다.
boolean isCommitted()	응답처리가 완료되였는가 안되였는가를 판단한다.
void reset()	완충기에 있는 자료를 삭제한다.
int getBufferSize()	완충기의 크기를 귀환한다.
String getCharacterEncoding()	의뢰기에 대한 응답에 해당하는 MIME자료를 보낼 때 사용하기 위해 현재 설정된 문자부호화를 얻는다.
ServletOutputStream getOutputStream()	의뢰기에 2진자료를 보내기 위해 사용하는 ServletOutputStream객체를 얻는다.
PrintWriter getWriter()	의뢰기에 본문자료를 보내기 위해 사용하는 PrintWriter 객체를 얻는다.
void setContentLength(int len)	응답으로서 의뢰기에 보내는 자료의 길이를 설정한다.
void setContentType(String type)	의뢰기에 대한 응답으로 보내는 자료의 형을 설정한다.
void setBufferSize(int size)	완충기의 크기를 설정한다.

ServletResponse는 HTTP규약전용의 응답을 처리할수 있는 대면이 아니다. 그러므로 ServletResponse을 확장하여 HttpServletResponse라는 대면을 만든다.

HttpServletResponse는 쿠키를 설정하거나 대화접속을 관리하는 메쏘드를 포함하고 있다.(표 4-9)

표 4-9. response객체의 주요 메쏘드들

Response객체의 주요 메쏘드	
public void addCookie(Cookie cookie)	주어진 쿠키를 응답에 추가한다. 즉 주어진 쿠키를 의뢰기가 저장하도록 HTTP응답머리부에 Cookie머리부를 추가한다.
public boolean containsHEADer(String name)	응답머리부에 주어진 이름의 항목이 있는 경우 true, 그 항목이 없는 경우 false를 귀환한다.
public String encodeRedirectURL(String url)	의뢰기와 봉사기사이에 대화접속이 유지되는 상태에서 열람기가 쿠키를 지원하지 않는 경우 주어진 URL를 sendRedirect메쏘드에서 사용하기 위하여 부호화한다.
public String encodeURL(String url)	주어진 URL에 대화접속 ID(식별자)를 포함하여 부호화한다.
public void sendError(int sc)	기능상 setStatus()메쏘드와 거의 같은데 주어진 상태코드와 그 코드에 해당하는 기정통보문을 사용하여 의뢰기에 오유를 보낸다.
public void sendError(int sc, String msg)	주어진 상태코드와 통보문을 사용하여 의뢰기에 오유를 보낸다.
public void sendRedirect(String location)	응답을 주어진 URL로 다시 전송한다. 참조파라메터 location은 절대 URL 혹은 상대적 URL로 지정한다. 이 메쏘드는 봉사기의 특정자원이 림시 다른 URL로 옮겨진 경우 사용하는 메쏘드이다.
public void setDateHEADer(String name, long date)	HTTP응답머리부의 날자를 주어진 이름과 날자를 가지도록 설정한다.
public void setHEADer(String name, String value)	주어진 이름과 값을 응답머리부항목에 추가한다.
public void setIntHEADer(String name, int value)	주어진 이름과 값을 가지도록 응답머리부항목을 옹근수형으로 설정한다.
public void setStatus(int sc)	전송될 HTTP응답의 상태코드를 설정한다.
public void setStatus(int sc, String sm)	응답을 위한 상태코드와 통보문을 설정한다.

제4절. Get방식과 Post방식의 동시처리

Get방식과 Post방식으로 의뢰기의 정보를 Servlet에 전달하고 전달된 정보를 출력하는 실례를 고찰하자.

우선 의뢰기의 웨브페지에서 홈양식으로 자료를 전달해야 한다. 이때 홈양식은 다음의 요소들을 포함해야 한다.

- action : 여기에서는 목표페지를 지정하여야 한다.
- method: 전송방식을 지정하여야 한다.
- 보내려는 자료: input(text, radio, hidden, textarea)에서 지정한다.
- 자료는 name을 포함하고 있어야 한다.

홈과 관련된 내용을 도식적으로 표현하면 그림 4-2와 같다. 그림에서 홈의 전송방식이 Post방식이라는것을 알수 있다. 그리고 action부분은 목표로 되는 URL을 나타내고있다.

《?age=20》는 URL에 붙여놓아 자동적으로 Get방식으로 전송된다. 즉 홈양식에서는 Post로 전송되고 action의 URL에서는 Get방식으로 전송하게 된다.

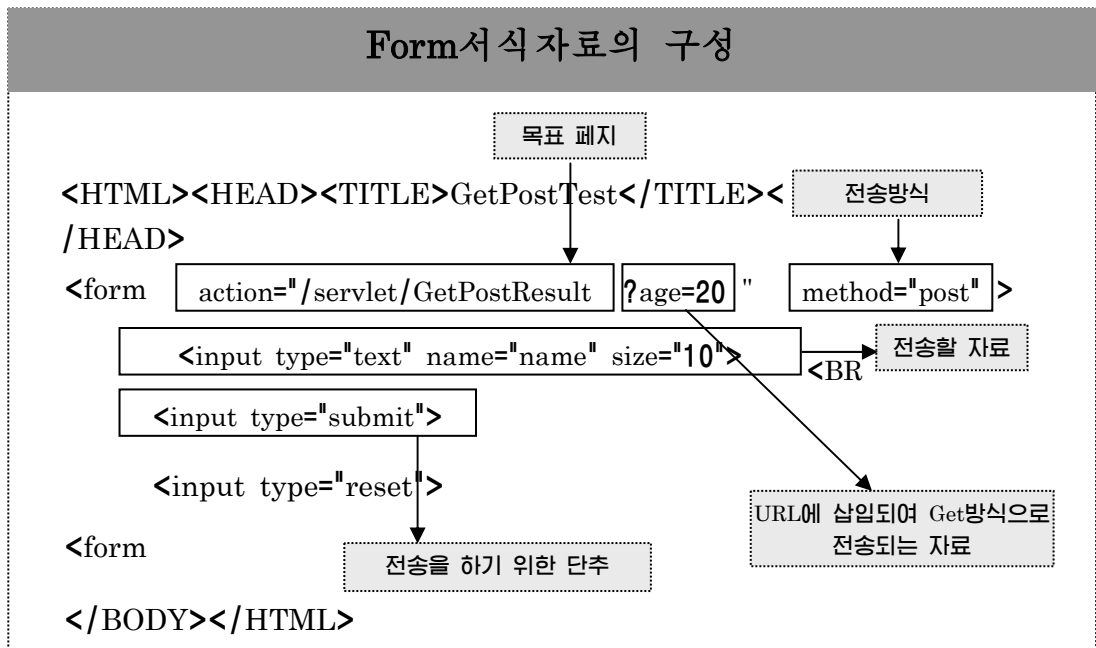



그림 4-2. Form에서 자료의 구성

우에서 도식적으로 나타낸것을 실제로 적용한 실례는 다음과 같다.

실례 4-1

GetPostTest.html

```
<HTML><HEAD><TITLE>GetPostTest</TITLE></HEAD><BODY>
<FORM action="/MySample/servlet/chap4.GetPostResult?age=20" method="Post">
  <input type="text" name="name" size="10"><BR>
  <input type="submit">
  <input type="reset">
</FORM></BODY></HTML>
```

그림 4-3. 실례 4-1의 실행결과

실례 4-2

GetPostResult.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetPostResult extends HttpServlet{
   public void doPost(HttpServletRequest req, HttpServletResponse res)
                                       throws ServletException, IOException{
      res.setContentType("text/html;charset=big5");
      PrintWriter pout=res.getWriter();
      String name=req.getParameter("name");
      String age=req.getParameter("age");
      pout.println("<HTML><HEAD>");
      pout.println("<TITLE>GetPostResult</TITLE>");
      pout.println("</HEAD><BODY>");
```

```
        pout.println("Name : "+name);
        pout.println("<BR>Age: "+age);
        pout.println("</BODY></HTML>");
    }
}
```

접근할 URL:

http://localhost:8080/MySample/servlet/chap4.GetPostResult?age=20

프로그람설명

HTML문서에서 age는 Get방식으로, name은 Post방식으로 자료를 전송한다. 그리고 의뢰기가 보낸 자료를 Servlet는 다음과 같은 코드를 리용하여 받는다.

String name=req.getParameter("name");

String age=req.getParameter("age");

HttpServletRequest의 getParameter메쏘드는 Get방식과 Post방식으로 전송된 자료를 변수이름에 넣는 메쏘드이다. 실행결과는 그림 4-4에서 보여주었다.

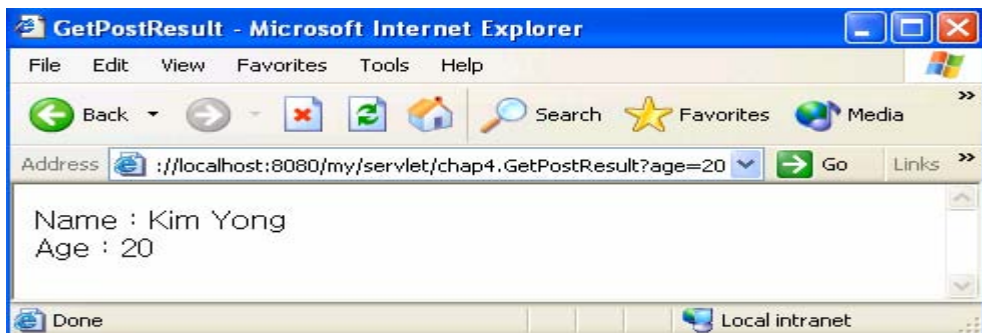



그림 4-4. 실례 4-2의 실행결과

제5절. Servlet에서 조선글처리원리

Java에서 조선글의 부호화는 유니코드방식을 사용한다. UTF-16부호화방식으로 2byte를 리용하여 하나의 문자를 표현한다. 문자를 바이트로 받아들이지 않으면 Java에서는 자동적으로 유니코드형식의 문자렬로 인식한다. Java에서 사용하는 조선글부호화방식은 유니코드방식이지만 웨브에서는 일반적으로 유니코드방식과는 다른 부호화방식을 사용하고있다. 웨브에서 사용하는 문자렬의 부호화방식은 Big5이다. 이것들은 사실 같은 방식이지만 미묘한 차이를 가지고있다.

Servlet에서 문자렬의 부호화는 복잡한 과정을 거쳐 진행된다.(그림 4-5)

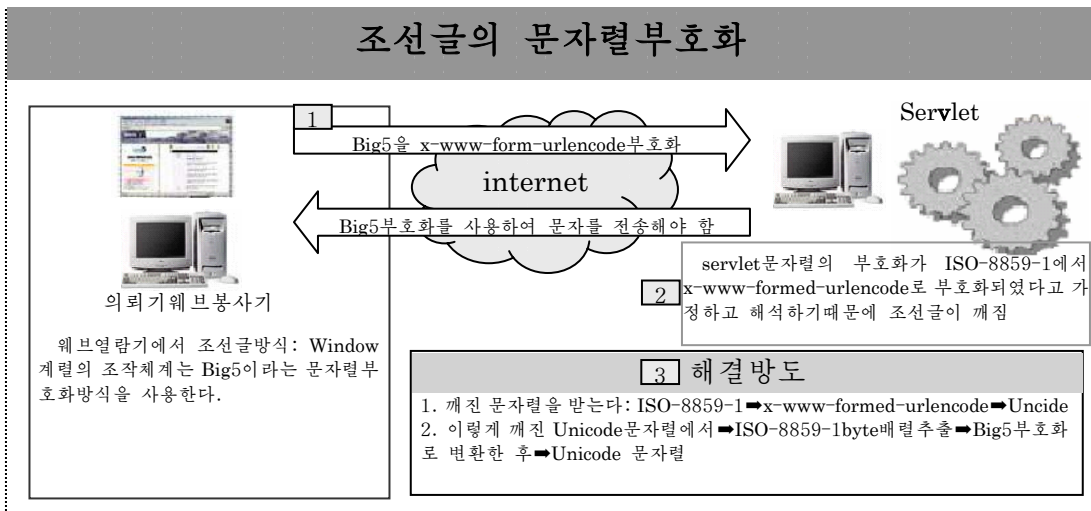



그림 4-5. 조선글의 문자렬부호화방식

웨브열람기의 조선글부호화방식은 Big5방식이다. 그리고 웨브열람기에서 자료가 전송될 때 웨브에서 통용되는 《x-www-form-urlencoded》형식으로 변환하여 전송하게 된다.

그러나 Servlet에서는 Big5로 웨브부호화되였다고 생각하지 않는다. Servlet는 라틴어 표준부호화방식인 ISO-8859-1방식으로 웨브부호화되였다고 보고 조선글을 받아들인다. 이렇게 하면 조선글이 깨지는 현상이 나타난다. 그것은 Big5로 부호화되였기때문이다.

Big5 → 웨브부호화 → 유니코드(조선글깨짐)

이때 해결방도는 깨여진 조선글에서 ISO-8859-1바이트묶음을 얻어내여 다시 Big5로 바꾸어주는 작업을 하여 유니코드로 변환하여야 한다.

유니코드(깨여진 조선글)→ ISO-8859-1 바이트묶음의 추출 → 원래 서식의 Big5로 변환

이렇게 하면 변환된 문자렬은 정확히 조선글이 되며 Java에서 유니코드형식으로 사용할수 있다.

여기서 한가지 주의해야 할것은 Servlet에서 조선글을 의뢰기로 전송하면 자동적으로 유니코드에서 ISO-8859-1형식으로 변환하여 전송한다는것이다. 그러므로 의뢰기가 웨브열람기를 사용하는 경우 조선글은 깨지게 된다. 이런 경우 웨브열람기가 알수 있는 Big5부호화방식으로 변환하여 전송해야 한다.

[주의]

Servlet에서 조선글을 전송할 때 Big5부호화를 사용하여 조선글의 깨짐을 방지한다.

설정방법: request.setContentType(“text/html;charset=big5”)

getParameter()메쏘드를 리용하여 의뢰기가 전송한 자료를 받았다면 우리는 응당 깨진 조선글이라고 생각해야 한다. 이렇게 받은 조선글은 ISO-8859-1부호화방식의 자료이므로 이것을 다시 Big5부호화방식의 자료로 변환해야 한다. 이것을 실현하자면 아래의 URigulEncoder클라스의 toBig5()메쏘드를 리용하면 된다. 그리고 반대로 big5부호화에서 ISO-8859-1부호화방식의 조선글을 얻자면 to8859_1()메쏘드를 리용하면 된다.

실례 4-3

URigulEncoder.java

```
package org.jabook.util;
public class URigulEncoder{
   public static String to 8859_1(String ko){
      if(ko==null){
         return null;
      }
      try{
         return new String(ko.getBytes("Big5"),"8859_1");
      }catch(Exception e){return ko;}
   }
   public static String toBig5(String en){
      if(en==null){
         return null;
      }
      try{
         return new String(en.getBytes("8859_1"),"Big5");
      }catch(Exception e){return en;}
   }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\org\jabook\util>javac URigulEncoder.java

프로그람설명

우의 클라스에서 사용한 2가지 기능을 보자. 우선 깨진 유니코드의 조선글에서 ISO-8859-1형식의 자료바이트를 추출하여 Big5부호화방식의 조선글로 변환한다.

웨브열람기에서 사용할수 있는 조선글로 만들어주는 방법은 다음과 같다.

new String(en.getBytes("8859_1"),"Big5");

다음은 그의 반대과정이다.

new String(ko.getBytes("Big5"),"8859_1");

우의 URigulEncoder클라스는 이 책의 전반에서 사용하는 클라스이므로 패키지형태로 만들어놓는다.

[주의]

콤파일할 때 -d추가선택으로 classes등록부에서 모든 Servlet파일을 콤파일하면 일없지만 만약 특정한 등록부에서 콤파일하면 URigulEncoder를 찾을수 없다. 이 경우에는 해당 classes등록부자체를 classpath에 추가하여야 한다.

제6절. HttpServletRequest클라스

4.6.1. 서식자료의 출력

의뢰기는 홈안의 input꼬리표를 리용하여 자료를 봉사기로 전송한다. 자료를 전송하는데 리용되는 입력형태에는 다음과 같은것이 있다.

- Text
- Hidden
- Radio
- Checkbox
- TextArea

다음은 홈을 통해 보낸 자료를 봉사기에서 얻는한 방법에 대하여 고찰한다.

실례 4-4

ParaNamesValueTest.html

```
<HTML><BODY>
<H1> 입력서식검증 </H1>
<Form action="/MySample/servlet/chap4.ParamsTest" method="Post">
1.본문마당 <input type=text name="tf1"><BR><BR>
2.Hidden <input type=hidden name="H1" value="감추어진  값"><BR><BR>
3.Radio단추 검증<BR>
<input type=radio name="r1" value="JSP" checked>JSP<BR>
<input type=radio name="r1" value="Java">Java<BR>
<input type=radio name="r1" value="Java스크립트">Java스크립트<BR><BR>
4.CheckBox단추 검증<BR>
<input type=CheckBox name="r2" value="평양">평양<BR>
<input type=CheckBox name="r2" value="남포">남포<BR>
<input type=CheckBox name="r2" value="신의주" checked>신의주<BR><BR>
5.select칸검증<BR>
<select name="s1">
 <option selected>김철수</option>
 <option>김상철</option>
 <option>안영희</option>
</select><BR><BR>
6.select칸 Multiple<BR>
<select name="s2" Multiple>
 <option selected>CGI</option>
 <option>JSP</option>
 <option>ASP</option>

 <option>Perl</option>
</select><BR><BR>
7.TextArea검증<BR>
<textarea name="ta1"></textarea> <BR>
<input type=submit value="전송">
</form>
</BODY></HTML>
```

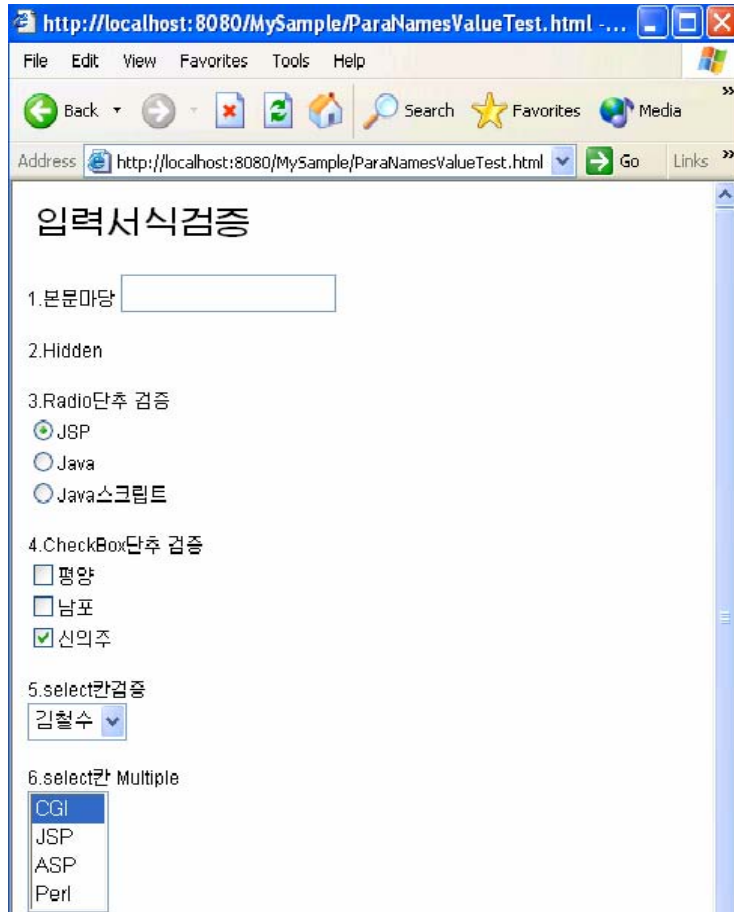



그림 4-6. 실례 4-4의 실행결과

프로그람설명

단일선택(radio)단추에서 하나만 선택하기 위하여서는 name에 같은 값을 주어야 한다.

```
<input type=radio name="r1" value="JSP" checked>JSP<BR>
<input type=radio name="r1" value="Java">Java<BR>
<input type=radio name="r1" value="Java스크립트">
```

검사칸(checkbox)단추에서도 name에는 같은 값을 입력하고 기정으로 검사칸이 선택된 상태로 하자면 checked속성을 리용한다.

```
<input type=CheckBox name="r2" value="평양" >평양<BR>
<input type=CheckBox name="r2" value="남포">남포<BR>
<input type=CheckBox name="r2" value="신의주" checked>신의주
```

select단추에서 multipul속성을 리용하면 목록안의 여러 항목을 동시에 선택할수 있다. 이때 Ctrl과 Shift건을 누른 상태에서 마우스단추를 찰칵하여 select안의 항목들을 여러개 선택하면 된다.

```
<select name="s2" Multiple>
```

```
<option checked>CGI</option>
<option>JSP</option>
<option>ASP</option>
<option>Perl</option>
</select>
```

그림 4-6은 이 실례의 실행결과이다.

Form끄리표의 action에 해당하는 ParamsTest클라스의 원천코드는 아래와 같다.

실례 4-5

ParamsTest.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.jabook.util.*;
public class ParamsTest extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse
            response) throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      String tf1 = request.getParameter("tf1");
      String h1 = request.getParameter("h1");
      String r1 = request.getParameter("r1");
      String[] r2 = request.getParameterValues("r2");
      String s1 = request.getParameter("s1");
      String[] s2 = request.getParameterValues("s2");
      String ta1 = request.getParameter("ta1");
      out.println("1.Text:"+encodeString(tf1)+"<BR>");
      out.println("2.Hidden:"+encodeString(H1)+"<BR>");
      out.println("3.Radio:"+encodeString(r1)+"<BR>");
      out.println("4:CheckBox:");
      for(int i=0; i<r2.length; i++)
         out.println(encodeString(r2[i])+"  ");
```

```
      out.println("<BR>5.Select:" + encodeString(s1) + "<BR>");
      out.println("6:Select Multipul:");
      for(int i=0; i<s2.length; i++)
         out.println(encodeString(s2[i]) +"  ");
      out.println("<BR>7.TextArea:" + encodeString(ta1) +"<BR>");
   }
   public String encodeString(String str){
      return URigulEncoder.toBig5(str);
   }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap4>javac ParamsTest.java
실행결과는 그림 4-7에서 보여주었다.

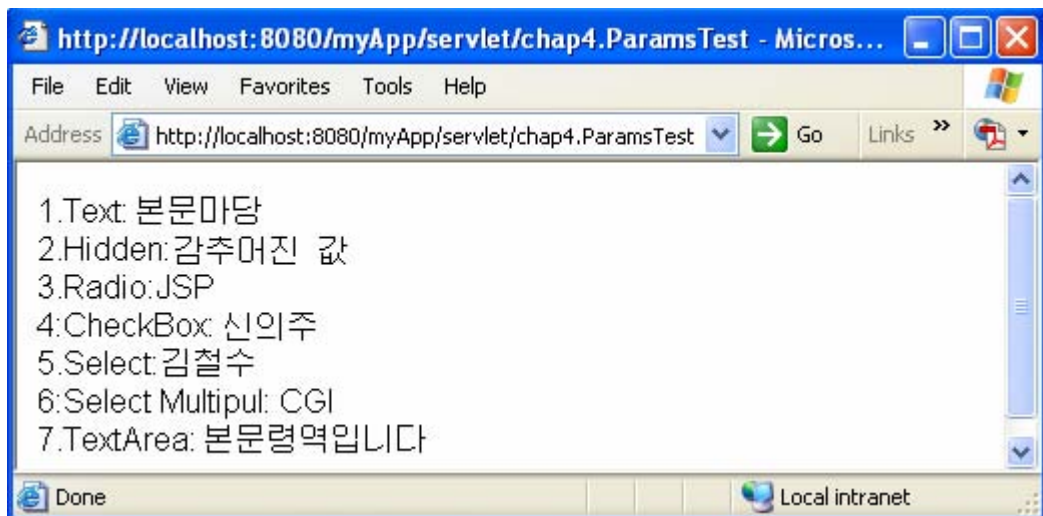



그림 4-7. 실례 4-5의 실행결과

프로그람설명

대부분의 입력된 자료는 HttpServletRequest의 getParameter()메쏘드로 값을 얻어낸다. 그러나 checkbox나 multipul, select와 같은 경우에는 값이 여러개 선택될수 있으므로 getParameterValues()메쏘드를 리용하여 값을 얻는다.

```
String tf1 = request.getParameter("tf1");
String H1 = request.getParameter("H1");
String r1 = request.getParameter("r1");
```

```
String[] r2 = request.getParameterValues("r2");
String s1 = request.getParameter("s1");
String[] s2 = request.getParameterValues("s2");
String ta1 = request.getParameter("ta1");
```

받은 자료가 조선글인 경우에는 출력할 때 깨져서 보이지 않으므로 조선글부호화과정이 필요하다. 그러므로 URigulEncoder클라스의 메쏘드를 리용하여 조선글부호화를 진행한다.

```
public String encodeString(String str){
  return URigulEncoder.toBig5(str);
}
```

자료를 부호화한 다음 출력흐름을 리용하여 자료를 출력한다.

```
out.println("1.Text:"+encodeString(tf1)+"<BR>");
out.println("2.Hidden:"+encodeString(H1)+"<BR>");
out.println("3.Radio:"+encodeString(r1)+"<BR>");
out.println("4:CheckBox:");
for(int i=0; i<r2.length; i++)
  out.println(encodeString(r2[i])+"  ");
out.println("<BR>5.Select:" + encodeString(s1) + "<BR>");
out.println("6:Select Multipul:");
for(int i=0; i<s2.length; i++)
  out.println(encodeString(s2[i]) +"  ");
out.println("<BR>7.TextArea:" + encodeString(ta1) +"<BR>");
}
```

for문에서 r2과 s2이 배렬형식으로 되여있으므로 자료를 모두 출력하게 된다.

4.6.2. getParameterNames()메쏘드

의뢰기는 Servlet를 Post방식으로 요청할 때 요청통보문의 본체부분을 getParameter()메쏘드를 리용하여 추출한다.

여기에서는 getParameterNames()메쏘드를 리용하여 모든 파라메터이름을 얻어내고 이것을 리용하여 파라메테의 값을 출력하는 실례를 고찰한다. 이 실례의 실행결과는 그림 4-8에서 보여주었다.

실례 4-6

RequestParameterNames.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.jabook.util.*;
public class RequestParameterNames extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      out.println("<HTML><BODY>");
      out.println("<H1>요청자료의 모든 서식자료받기</H1>");
      Enumeration enum = request.getParameterNames();
      while(enum.hasMoreElements()){
          String name = (String)enum.nextElement();
          out.println("<b>"+name+":</b>"+
          URigulEncoder.toBig5(request.getParameter(name))+"<BR>");
      }
      out.println("</BODY></HTML>");
      out.close();
   }
}
```

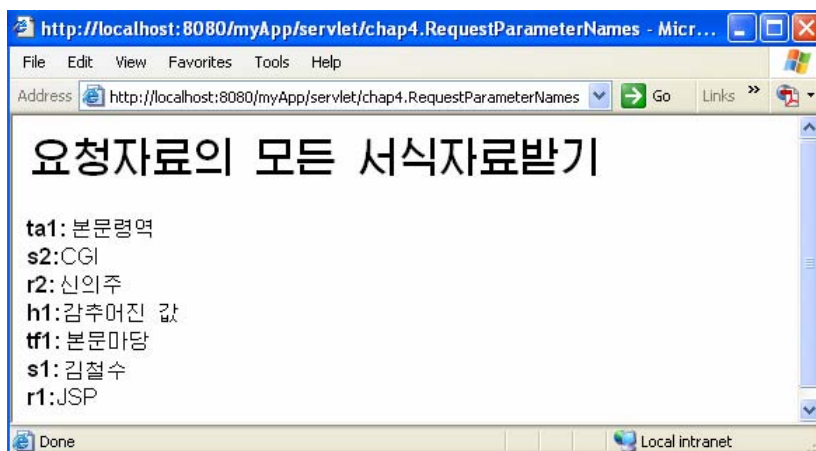



그림 4-8. 실례 4-6의 실행결과

프로그람설명

getParameterNames()메쏘드를 리용하여 모든 파라메터이름을 얻어낸다.

Enumeration enum = request.getParameterNames();

getParameter()메쏘드를 리용하여 enum안에 들어있는 매개 파라메터이름들에 해당하는 값을 얻는다. 그리고 그 값들의 조선글회복을 위하여 URigulEncoder.toBig5()메쏘드를 호출하고있다.

String name = (String)enum.nextElement();

URigulEncoder.toBig5(request.getParameter(name))

일반적으로 파라메터이름을 안다면 직접 getParameter()메쏘드를 리용할수 있지만 모른다면 getParameterNames메쏘드를 리용하여 모든 Parameter Name을 얻은 후 그 값을 얻어야 한다.

4.6.3. 머리부의 출력

HTTP규약의 통보문은 크게 세부분 즉 요청(request)행부분과 머리부(Header)부분 그리고 본체(entity)부분으로 나누어진다.

그림 4-9는 HTTP규약의 통보문을 구조적으로 분석한 그림이다.

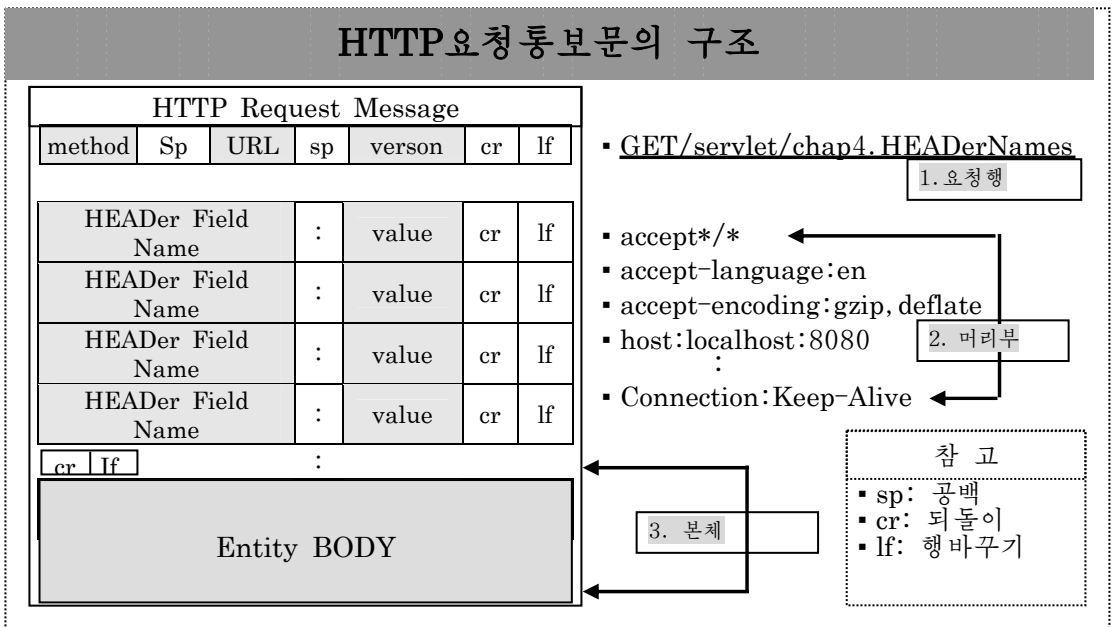



그림 4-9. 요청통보문의 구조

- **요청행부분**

HTTP요청행통보문에는 method, URL, version이 포함된다.

매개는 공백(space)으로 구분되며 제일 마지막에는 《\r》(되돌이)나 《\n》(행바꾸기)를 추가한다. 실례로

GET /servlet/chap4.HEADerNames HTTP/1.1

- **머리부행부분**

머리부행부분은 여러 줄의 《HEADer field name : value》형식으로 되여있다. 구분기호로서는 《:》을 사용하고있으며 마지막에는 요청행부분과 마찬가지로《\r》나 《\n》를 포함하고있다.

accept:*/*

accept-language:en

accept-encoding:gzip, deflate

host:localhost:8080

우의 그림에 있는 머리부외에도 여러가지 머리부들이 존재한다. 자주 보게 되는 머리부이름들을 아래에 소개한다.

Accept, Accep-Charset, Accept-Encoding, Accept-Lanaguage, Authorization, Chache-Control, Connection, Content-Type, Cookie, Expect, From, Host, If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since, Pragma, Proxy-Authorization, Range, Referer, Upgrade, User-Agent, Via, Warning

머리부를 추출하는 실례를 보기로 하자. 머리부를 추출하기 위하여 getHEADerNames()메쏘드를 리용하여 머리부에 있는 이름을 전부 추출하고 getHEADer()메쏘드를 리용하여 머리부값들을 추출하고있다.

실례 4-7

HeaderNames.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HeaderNames extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<HTML><BODY>");
        Enumeration enum = request.getHeaderNames();
        while(enum.hasMoreElements()){
```

```
            String name = (String)enum.nextElement();
            String value = request.getHeader(name);
            out.println( name + ":" + value +"<BR>");
        }
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

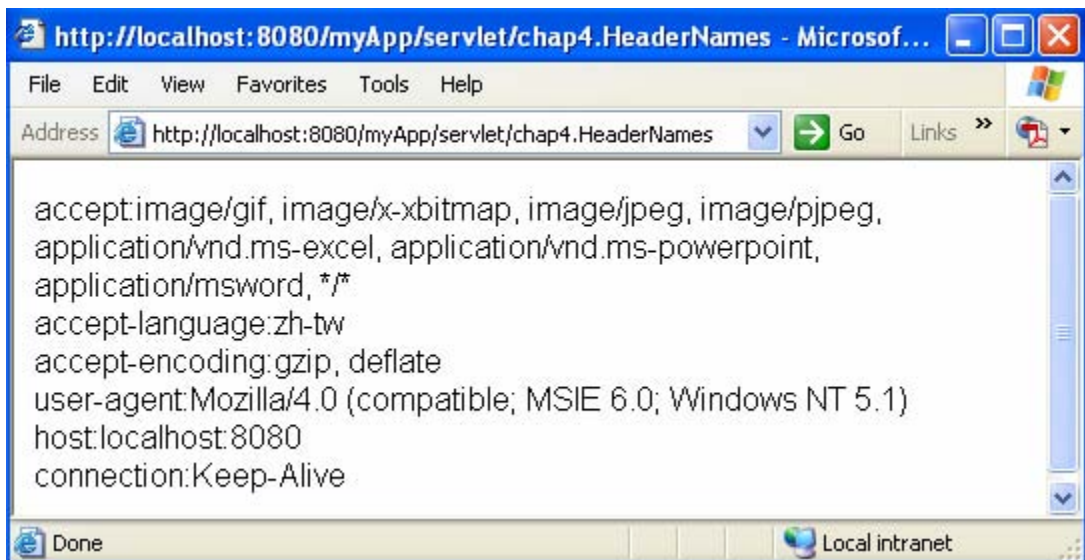



그림 4-10. 실례 4-7의 실행결과

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap4>javac HEADerNames.java
실행결과는 그림 4-10과 같다.

프로그람설명

머리부의 모든 머리부이름을 추출하기 위하여 다음의 명령문을 사용한다.

```
Enumeration enum = request.getHeaderNames();
```

머리부이름을 Enumeration형으로 얻어내여 순환하면서 그 값을 추출한다.

```
while(enum.hasMoreElements()){
String name = (String)enum.nextElement();
String value = request.getHeader(name);
out.println( "<b>"+name+"</b>"+":"+value +"<BR>");
}
```

값을 추출하기 위하여 getHeader(name)메쏘드를 사용하였다.

4.6.4. 메쏘드를 리용한 머리부의 출력

앞에서 getHeaderNames()메쏘드를 리용하여 모든 머리부를 출력하는 방법에 대하여 보았다. 여기에서는 매개 머리부를 HttpServletRequest의 메쏘드를 리용하여 얻어내는 방법에 대하여 보기로 한다.

머리부정보를 추출할수 있는 HttpServletRequest의 메쏘드에는 다음과 같은것이 있다(표 4-10).

표 4-10. HttpServletRequest의 메쏘드들

머리부에 포함된 매 요소들의 내용을 출력하는 메쏘드	
getAuthType, getRemoteUser	Authorization머리부에 담긴 내용들가운데서 해당한 내용만을 얻을 때 리용한다. 이것은 인증과 관련된 부분이다.
getContentLength	내용의 길이를 얻어내는 메쏘드이다.
getContentType	내용의 자료형을 얻어내는 메쏘드이다.
getMethod	의뢰기의 요청방식을 얻어내는 메쏘드이다.
getRequestURI	URL에서 주콤퓨터와 포구 다음부터 서식자료이전까지의 부분을 얻어내는 메쏘드이다.
getProtocol	의뢰기의 요청행에서 판본을 얻어내는 메쏘드이다.
첨부된 모든 머리부를 출력하는 메쏘드	
getHeaderNames	의뢰기의 요청통보문에서 사용된 모든 머리부이름을 얻어내는 메쏘드이다.
getHeader	머리부이름으로 머리부값을 얻어내는 메쏘드이다.

다음의 실례는 간단한 홈의 자료를 Post방식으로 Servlet에 보내서 Servlet에서 머리부를 출력하는 실례이다. 실행결과는 그림 4-11에서 보여준다.

실례 4-8

HEADerMethodTst.html

```
<HTML><BODY>
<H1> 메쏘드를 리용한 머리부추출 </H1>
<Form action="/MySample/servlet/chap4.HEADerMethod"
     method="Post" type="text/html">
    통보문 <input type=text name="tf"><BR><BR>
    <input type=submit value="전송">
</Form>
</BODY></HTML>
```

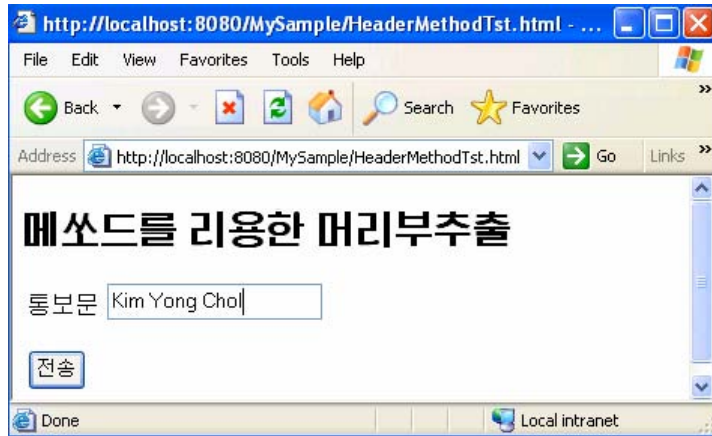



그림 4-11. 실례 4-8의 실행결과

실례 4-9

HeaderMethod.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HeaderMethod extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("<HTML><BODY>");
    out.println("<H1>" + request.getParameter("tf") + "</H1>");
    out.println("<b>getAuthType()</b>: " + request.getAuthType()+"<BR>");
    out.println("<b>getContentType()</b>: " + request.getContentType()+"<BR>");
    out.println("<b>getContentLength()</b>: " + request.getContentLength()+"<BR>");
    out.println("<b>getMethod()</b>: " + request.getMethod()+"<BR>");
    out.println("<b>getRequestURI()</b>: " + request.getRequestURI()+"<BR>");
    out.println("<b>getProtocol()</b>: " + request.getProtocol()+"<BR>");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap4>javac HeaderMethod.java

실행결과는 그림 4-12에서 보여준다.

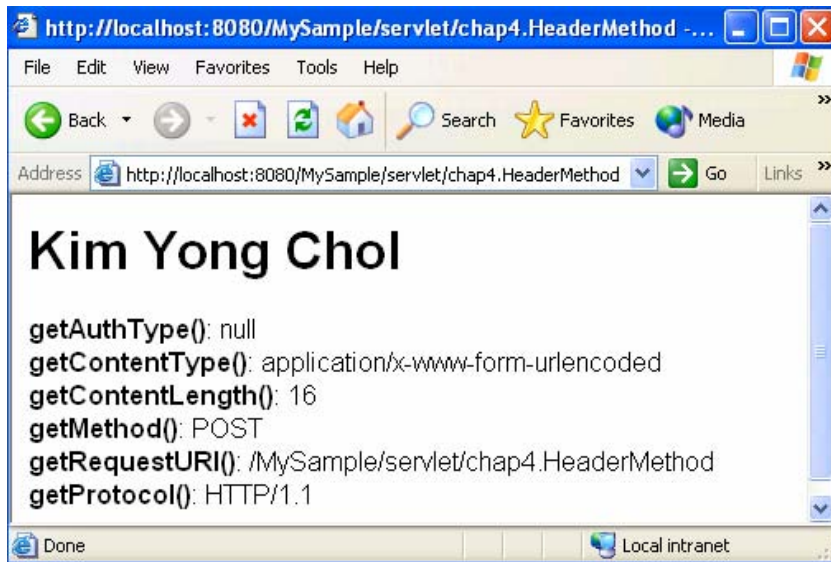



그림 4-12. 실례 4-9의 실행결과

프로그람설명

현재의 요청에는 인증에 관한 설정이 없으므로 getAuthType가 null로 된다. 그리고 Post 방식으로 요청이 이루어질 때 내용의 자료형과 길이가 우와 같이 나타난다. getRequestURI()메쏘드는 요청 URL에서 주콤퓨터와 포구(localhost:8080)다음부터 서식자료이전까지의 주소내용을 얻어내는 메쏘드이다. 즉

URL에서 《MySample/servlet/chap4.HeaderMethod》만을 얻어낸다.

getProtocol은 Request요청행에서 Version부분을 얻어내는 메쏘드이다.

실례에서 Request요청행은 다음과 같다.

GET/MySample/servlet/chap4.HeaderMethod HTTP/1.1

여기서 이 메쏘드를 리용하면 HTTP/1.1이 얻어진다.

4.6.5. CGI변수와 Servlet메쏘드

의뢰기의 요청에 대한 모든 정보를 CGI(공통관문대면부)프로그람에 알려주기 위한 수단으로 CGI변수를 사용한다. Servlet도 CGI프로그람의 한 종류이므로 의뢰기의 요청에 관한 정보를 얻는데 CGI변수를 리용한다.

일반적으로 CGI변수들의 구성은 다음과 같다.

· HTTP Request행에서 추출된 정보
· HTTP Header에서 추출된 정보
· HTTP Connection에 대한 정보
· 봉사기설정자체에 대한 정보

Servlet에서는 아래와 같은 CGI변수에 해당하는 메쏘드들을 포함하고있다.

```
AUTH_TYPE → request.getAuthType()
CONTENT_LENGTH → request.getContentLength()
CONTENT_TYPE → request.getContentType
DOCUMENT_ROOT → getServletContext().getRealPath( “/” );
PATH_INFO → request.getPathInfo()
PATH_TRANSLATED → request.getPathTranslated()
QUERY_STRING → request.getQueryString()
REMOTE_ADDR → request.getRemoteAddr()
REMOTE_HOST → request.getRemoteHost()
REMOTE_USER → request.getRemoteUser()
REQUEST_METHOD → request.getMethod()
SCRIPT_NAME → request.getServletPath()
SERVER_NAME → request.getServerName()
SERVER_PORT → request.getServerPort()
SERVER_PROTOCOL → request.getProtocol()
SERVER_SOFTWARE → getServletContext().getServerInfo()
```

실례 4-10

CGIVariable.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CGIVariable extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.println("<HTML><BODY>");
      out.println("<H1>CGI Variable & Servlet Methods</H1>");
      out.println("<b>AUTH_TYPE()</b>: " + request.getAuthType()+"<BR>");
      out.println("<b>CONTENT_LENGTH()</b>: "+request.getContentLength()+"<BR>");
      out.println("<b>CONTENT_TYPE()</b>: "+request.getContentType()+"<BR>");
```

```
    out.println("<b>DOCUMENT_ROOT()</b>: " +
                      getServletContext().getRealPath("/")+"<BR>");
    out.println("<b>PATH_INFO()</b>: " + request.getPathInfo()+"<BR>");
    out.println("<b>PATH_TRANSLATED()</b>: " + request.getRemoteHost()+"<BR>");
    out.println("<b>REMOTE_USER()</b>: " + request.getRemoteUser()+"<BR>");
    out.println("<b>REQUEST_METHOD()</b>: " + request.getMethod()+"<BR>");
    out.println("<b>스크립트_NAME()</b>: " + request.getServletPath()+"<BR>");
    out.println("<b>SERVER_NAME()</b>: " +
                      request.getServerName()+"<BR>");
    out.println("<b>SERVER_PORT()</b>: " + request.getServerPort()+"<BR>");
    out.println("<b>SERVER_PROTOCOL()</b>: " +
                      request.getProtocol()+"<BR>");
    out.println("<b>SERVER_SOFTWARE()</b>: " +
                      getServletContext().getServerInfo()+"<BR>");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap4>javac CGIVariable.java

그림 4-13은 이 실례의 실행결과이다.

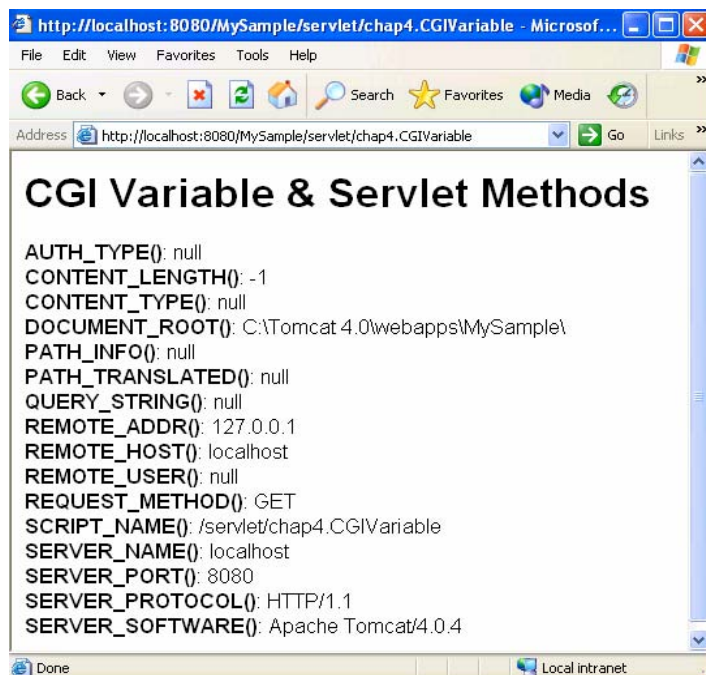



그림 4-13. 실례 4-10의 실행결과

다음과 같은 주소로 접속해본다.
http://localhost:8080/MySample/servlet/chap4.CGIVariable

4.6.6. Request통보문을 보내여 페지를 받기

의뢰기가 통보문을 요청하고 HTTP봉사기가 처리하는 과정은 몇단계의 절차를 거친다. 의뢰기가 통보문을 요청할 때 HTTP봉사기가 처리하는 과정은 아래의 그림과 같다(그림 4-14).

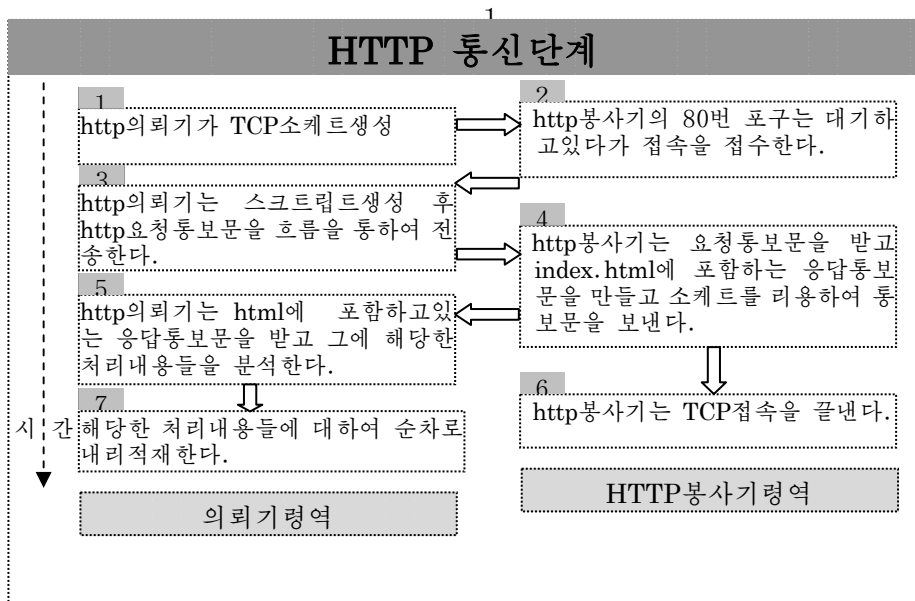



그림 4-14. HTTP통신단계

먼저 요청통보문을 만들어야 한다. 그리고 요청통보문을 HTTP웨브봉사기로 보낸다. 이것을 프로그람을 실행하여 보기로 하자.

실례 4-11

RequestMessage.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
public class RequestMessage extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
      OutputStream out = response.getOutputStream();
```

```
        InetAddress webServer = InetAddress.getByName("localhost");
        Socket httpPipe = new Socket(webServer, 80);  //웨브봉사기포구:80
        InputStream is = httpPipe.getInputStream();
        PrintStream os = new PrintStream(httpPipe.getOutputStream());
        os.println("GET " + "/index.html" + " HTTP/1.0\n"); // GET HTTP
        int temp;
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        while ( (temp = is.read())!=-1) {
           baos.write(temp);
        }
        out.write(baos.toByteArray());
        os.close();
        is.close();
        out.close();
    }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap4>javac RequestMessage.java

실행결과는 그림 4-15와 같다.

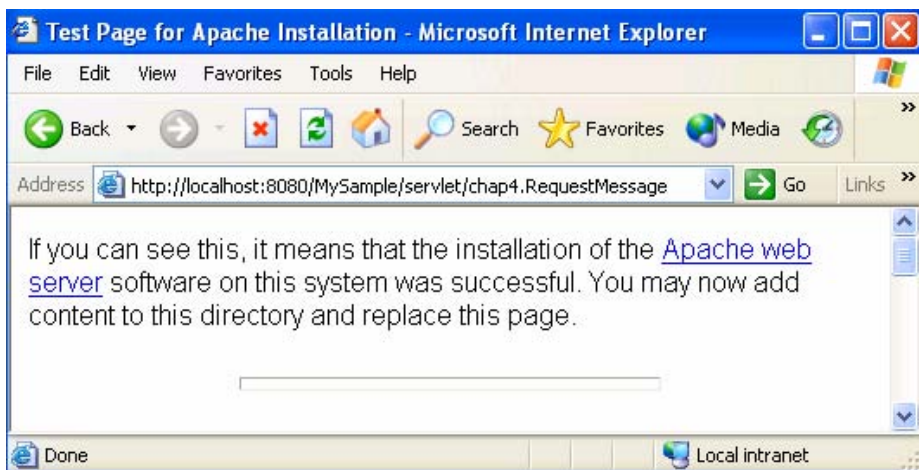



그림 4-15. 실례 4-11의 실행결과

프로그람설명

먼저 통신하기 위한 소케트를 생성한다. 현재 련결하려고 하는 봉사기는 《http://localhost:8080/index.html》이다. URL를 리용하여 소케트를 생성하는 방법은 다음과 같다.

```
InetAddress webServer = InetAddress.getByName("localhost");
Socket httpPipe = new Socket(webServer, 80);//웨브봉사기포구:80
```

소케트가 생성되면 해당 소케트에 입력흐름과 출력흐름을 생성하여 해당 웨브봉사기로 자료를 보내거나 의뢰기가 자료를 받을수 있다. 다음과 같은 명령문으로 생성된 소케트에서 흐름을 연다.

```
InputStream is = httpPipe.getInputStream();
PrintStream os = new PrintStream(httpPipe.getOutputStream());
```

흐름이 생성되면 출력흐름을 리용하여 련결된 웨브봉사기에 요청통보문을 보낸다. 요청통보문을 만드는 방법은 앞에서 고찰하였다. 주의할것은 요청통보문을 구성할 때 공백으로 구분하여야 한다는것이다.

```
os.println("GET "+ "/index.html " + "HTTP/1.0\n ");
```

출력흐름으로 Request통보문을 보내면 소케트에 련결된 입력흐름으로 자료가 들어오게 된다. 자료를 ByteArrayOutputStream에 적재하고 적재된 바이트들을 Servlet의 out흐름으로 출력하여 보낸다.

```
int temp;
ByteArrayOutputStream baos = new ByteArrayOutputStream();
while ( (temp = is.read())!=-1) {
baos.write(temp);
}
out.write(baos.toByteArray());
```

의뢰기가 전송하는 응답통보문의 구성도는 다음과 같다(그림 4-16).

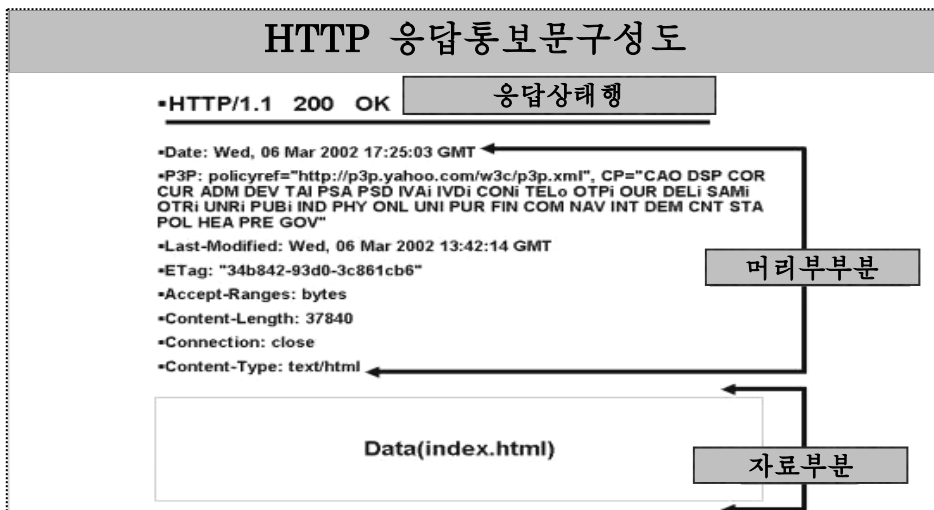



그림 4-16. HTTP응답통보문의 구성도

• 응답통보문의 구성요소

응답통보문은 응답상태행(Response Status Line)과 머리부부분(HEADer Lines) 그리고 자료부분(data)으로 나누어진다.

응답상태행에는 판본과 상태를 나타내는 내용이 들어간다.

HTTP/1.1 200 OK

현재 200 OK를 표시하고 있지만 여러가지 상태에 따라 번호가 다르다.상태번호에는 다음과 같은것이 있다(표 4-11).

표 4-11. 상태에 따르는 상태번호들

상태번호	설 명
200 OK	의뢰기의 요청이 성공적으로 진행된다.
301 Moved Permanently	요청 URL이 이동되고 새로운 위치가 정해진다.
400 Bad Request	봉사기가 리해할수 없는 요청통보문이다.
404 Not Found	해당 웨브봉사기에서 요청 URL을 찾을수 없다.
505 HTTP Version Not Supported	현재의 판본에서는 지원하지 않는다.

다음으로 머리부부분은 응답에 대한 머리부이름과 값으로 구성되여있으며 응답에 필요한 머리부들을 포함하고있다. 아래에서는 응답의 머리부부분의 출력내용들을 보여주고 있다.

```
Date: Wed, 06 Mar 2002 17:25:03 GMT
P3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAi IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA PRE GOV"
Last-Modified: Wed, 06 Mar 2002 13:42:14 GMT
ETag: "34b842-93d0-3c861cb6"
Accept-Ranges: bytes
Content-Length: 37840
Connection: close
Content-Type: text/html
```

제7절. HttpServletResponse클라스

4.7.1. 다른 페지로 이동

작업을 수행하다가 작업의 결과에 따라서 또는 필요에 따라서 다른 페지로 이동해야 할 경우가 있다. 이때 HttpServletResponse의 sendRedirect메쏘드를 리용한다. sendRedirect의 형식을 보면 다음과 같다.

public void sendRedirect(java.lang.String location) throws java.io.IOException

아래의 실례는 sendRedirect메쏘드를 리용하여 다른 페지로 이동하는 실례이다. doGet메쏘드를 리용하여 URL로부터 Tomcat라는 값을 받으면 《http://localhost:8080/》 페지로 이동한다.

실례 4-12

ResponseRedirect.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ReponseRedirect extends HttpServlet {
   public void doGet(HttpServletRequest request,  HttpServletResponse response)
                  throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      String gopage = request.getParameter("gopage");
      if(gopage!=null && gopage.equals("Tomcat")){
         response.sendRedirect("http://localhost:8080/");
      }else {
         out.println("<HTML><BODY>");
         out.println("<H1>Response의 sendRedirect</H1>");
         out.println("다른페지로 이동하기 위하여서는<BR>");
         out.println("URL에 ?gopage=Tomcat를 붙이십시오");
         out.println("</BODY></HTML>");
      }
      out.close();
   }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap4>javac ReponseRedirect.java

프로그람설명

열람기의 주소칸에 gopage=Tomcat를 입력하고 Enter건을 누르면 sendRedirect메쏘드에 지적된 페지로 이동한다. 즉

response.sendRedirect("http://localhost:8080/");

sendRedirect는 단순히 다른 페지로 이동할 때 리용한다. 프로그람에서 어디에 위치하든지 호출된 시점에서 원하는 다른 페지로 이동하며 현재 진행하던 작업도 호출되기 직전까지 모두 수행된 후 페지가 이동된다. 그림 4-17에서 이 실례의 실행결과를 보여준다.

그림 4-17. 실레 4-12의 실행결과

4.7.2. Response의 상태

HttpServletResponse에는 응답의 상태를 나타내는 성원마당들을 포함하고있다. HTTP규약에는 응답을 표시하는 많은 상태들이 존재한다. 가장 많이 사용되는 마당은 다음과 같다.

SC_OK(200) → OK SC_BAD_REQUEST(400) → 잘못된 요청 SC_FORBIDDEN(403) → 권한 없음

이와 같이 응답과 관련된 마당들은 Servlet에서 의뢰기로 응답을 보낼 때 사용한다. 다음의 실례는 임의로 응답들을 의뢰기에게 전송하는 방법을 보여주기 위한 실례이다. 이 실례의 실행결과들은 그림 4-18에서 보여주었다.

실례 4-13

ResponseStatus.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ReponseStatus extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      String temp = request.getParameter("status");
      int status;
      if(temp != null){
         status = Integer.parseInt(temp);
         if(status==400){
            response.sendError(HttpServletResponse.SC_BAD_REQUEST);//1 잘못된 요청
         }else if(status==403) {
            response.sendError(HttpServletResponse.SC_FORBIDDEN);//2 권한 없음
         }
      }else{
         PrintWriter out = response.getWriter();
         out.println("<HTML><BODY>");
         out.println("<H1>Response의 응답상태</H1>");
         out.println("URL 뒤에 ?status=400, status=403");
         out.println("을 붙여서 요청하십시오");
         out.println("</BODY></HTML>");
         out.close();
      }
   }
}
```

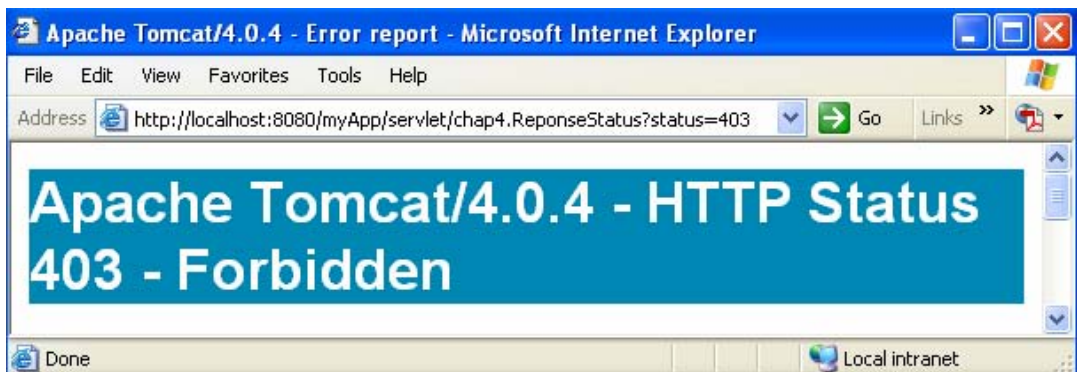
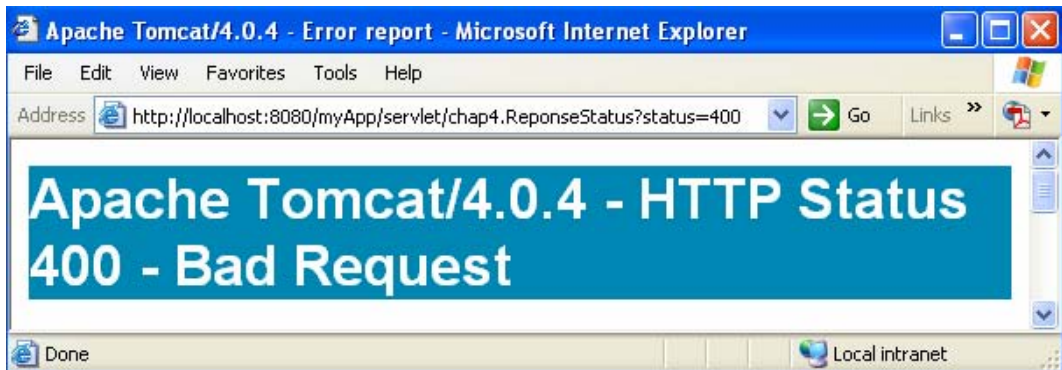
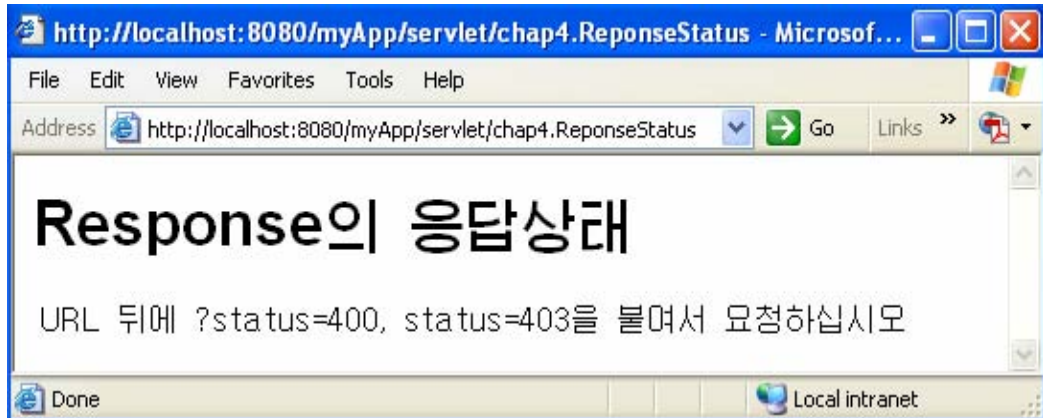



그림 4-18. 실례 4-13의 실행결과

프로그람설명

결과를 보면 머리부의 상태를 400과 403으로 설정함에 따라 그에 따르는 통보문이 각각 《Bad Request》, 《Forbidden》이라고 나타나는것을 확인할수 있다. 다음 절에서 이런 상태와 머리부를 설정하는 방법에 대하여 보다 자세히 고찰한다.

4.7.3. setHeader메쏘드

주어진 이름과 값으로 머리부를 설정할 때 일반적으로 쓰이는것은 HttpServletResponse의 setHeader메쏘드인데 그 형식은 다음과 같다.

public void setHeader(java.lang.String name, java.lang.String value)

여기서는 유용하게 쓰이는 몇가지 머리부를 간단히 소개한다(표 4-12).

표 4-12. 주요한 몇가지 머리부

머리부	설명
Cache-Control	의뢰기가 받은 문서를 고속완충기억기에 저장할 여부를 설정하는것으로서 그 제한정도에 따라서 public, private, no-cache, no-store, must-revalidate 등의 값을 가진다.
Connection	열람기의 련결방식을 지정한다. 열람기가 영구적으로 HTTP련결을 사용하지 말아야 한다면 close값으로 지정하면 된다. 기정으로는 영구적인 련결로 설정되여있다.
Expires	고속완충기억기에 문서가 남아있는 시간을 설정한다.
Refresh	열람기가 자동적으로 적재되게 한다.
Set-Cookie	쿠키를 설정하는데 쓰인다.

아래의 실례는 setHeader의 간단한 사용법을 보여주는 실례이다. setHeader메쏘드에서 Refresh속성을 설정하여 현재 페지에서 5s후에 다른 페지로 이동하도록 한다.

실례 4-14

SetHeaderTest.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SetHeaderTest extends HttpServlet{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws IOException, ServletException{
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      response.setHeader("Refresh", "5;URL=http://locahost");
      out.println("<HTML><BODY>");
      out.println("<H3>잠간만 기다려 주십시오... </H3>");
      out.println("<H2> 5초후에 다음  홈페지로 이동합니다.</H2>");
      out.println("</BODY></HTML>");
   }
}
```

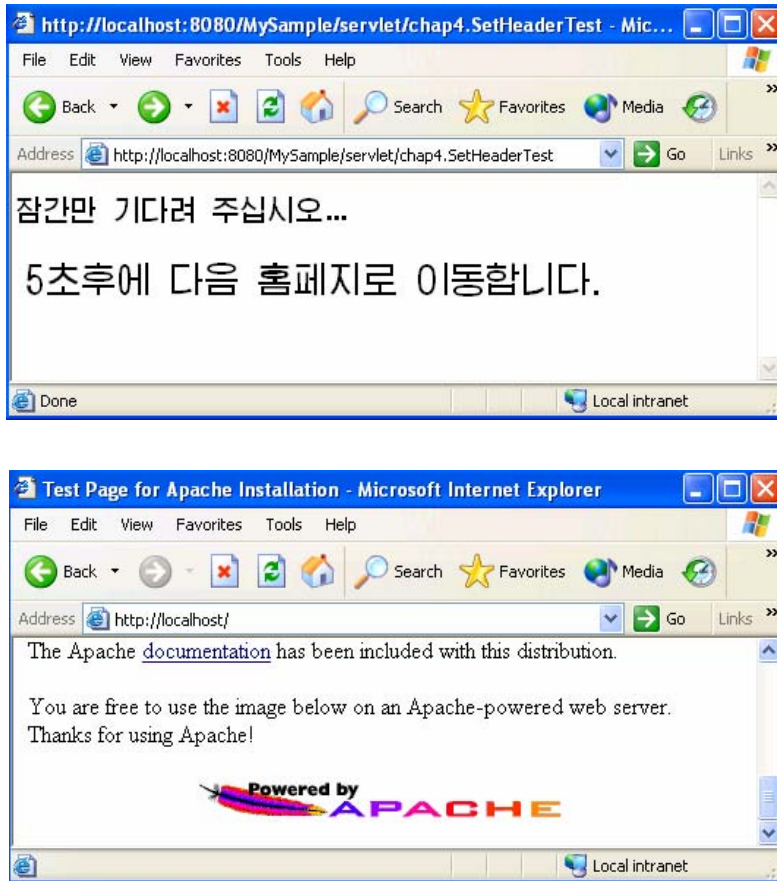



그림 4-19. 실례 4-14의 실행결과

프로그람설명

프로그람을 실행시키면 우의 그림과 같이 Servlet페지가 웨브열람기에 현시되는데 5s후에 설정해놓은 《http://localhost》로 이동한다. 이것은 setHeader메쏘드에 다음과 같은 설정을 진행하면 된다.

response.setHeader("Refresh","5;URL=http://localhost");

그리고 머리부를 설정하기 전에 어떤 머리부가 이미 설정되였는가를 알려면 containsHeader메쏘드를 리용하면 된다. 실행결과는 그림 4-19와 같다.

4.7.4. 인증페지 만들기

보통 웨브봉사기프로그람작성에서 등록가입하는데 필요한 인증페지를 만들 때 HTML서식을 많이 리용한다. 그러나 어느 거점에 들어가려 할 때 등록가입을 위한 인증페지가 웨브열람기와는 다른 별개의 창으로 떠서 사용자식별자와 통과암호를 요구하는 경우에는 봉사기측에서 Http를 기반으로 하여 만든다.

여기에서는 Servlet에서 Http를 기반으로 하는 기본인증(basic Authorization)방법에 대하여 고찰한다.

먼저 Servlet에서 기본인증(basic Authorization)을 리용하는 절차를 고찰한다.

- 인증정보를 담고있는 Authorization머리부의 존재여부를 우선 검사한다.
- 머리부가 존재하는 경우 《basic》란 단어를 뺀 나머지 부분을 base64로 복호화한다. 인증방식을 나타내는 《basic》란 단어 뒤부분부터 실제적인 사용자의 정보가 시작된다. 여기서 base64는 간단한 암호화알고리듬이다. Sun회사에서는 이렇게 base64로 부호화된 문자렬을 복호화하기 위한 sun.misc.BASE64클라스를 제공하고있다.
- 복호화한 후 생긴 《user:password》형태의 문자렬로 인증조작을 진행한다. base64로 복호화한 후 나오는 사용자정보(인증을 위해 입력한 정보)를 자료기지 등에 입력된 실제 정보와 비교하여 일치하면 원하는 페지에 들어가게 되고 그렇지 않은 경우 다음의 과정을 거친다.

Authorization머리부가 존재하지 않거나 인증이 실패하는 경우 401상태코드와 머리부를 되돌려준다.

Authorization머리부가 존재하지 않거나 입력된 사용자의 정보가 잘못되여 인증조작에서 실패한 경우는 사용자식별자와 통과암호를 문의하는 창이 나타나는데 그것을 해주는것이 401상태코드(Unauthorized)와 《www-Authentificate: BASIC realm="name"》형태의 머리부이다.

상태코드는 Http응답의 맨 웃줄에 상수와 간단한 통보문으로 구성되는데 매 번호에 따라서 의뢰기에게 정보를 제공하거나 처리의 상태, 파일의 이동, 오유상태 등을 나타낸다.

여기서 401(SC_UNAUTHORIZED)은 의뢰기가 Authorization머리부에 틀린 정보를 넣고 특정한 페지에 접근하려 한다는것을 의미한다. 그리고 www-Authentificate머리부는 401상태코드 설정시 반드시 응답으로 포함시켜야 하는것으로써 Authorization요청머리부에 포함시킬 인증형식 및 령역을 열람기에 알려준다. 우에서 보면 인증형식은 BASIC이고 령역은 name인 사용자식별자와 통과암호를 입력할 대화창을 현시한다.

- 사용자식별자와 통과암호를 입력한 후 그것을 다시 base64로 부호화하고 Authorization머리부에 넣어 요청을 보낸다.

다음의 실례는 우의 인증절차를 구체적으로 실현하는 실례로서 사용자식별자와 통과암호를 검사하여 입력이 정확할 때에만 원하는 페지에 접근할수 있게 한다.

실례 4-15

PassCheck..java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import sun.misc.BASE64Decoder;
public class PassCheck extends HttpServlet {
   public void init(ServletConfig config) throws ServletException {
      super.init(config);
   }
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      String authorization = request.getHeader("Authorization");
      if (authorization == null) {
         response.setStatus(response.SC_UNAUTHORIZED);
         response.setHeader("WWW-Authenticate", "BASIC realm=\"novel JSP-Servlet\"");
      }else{
         String userInfo = authorization.substring(6).trim();
         BASE64Decoder decoder = new BASE64Decoder();
         String info = new String(decoder.decodeBuffer(userInfo));
         int index = info.indexOf(":");
         String user = info.substring(0, index);
         String password = info.substring(index+1);
         String realUser = "Admin";
         String realPassword = "1234";
         if (realUser.equals(user) && realPassword.equals(password)) {
            out.println("<HTML><BODY>");
            out.println("<H3>인증성공</H3>");
            out.println("<H2>Novel-Servlet에 오신것을 환영합니다.</H2>");
            out.println("</BODY></HTML>");
         }else{
            response.setStatus(response.SC_UNAUTHORIZED);
            response.setHEADer("WWW-Authenticate", "BASIC realm=
                  \"novel JSP-Servlet\"");
         }
      }
   }
}
```

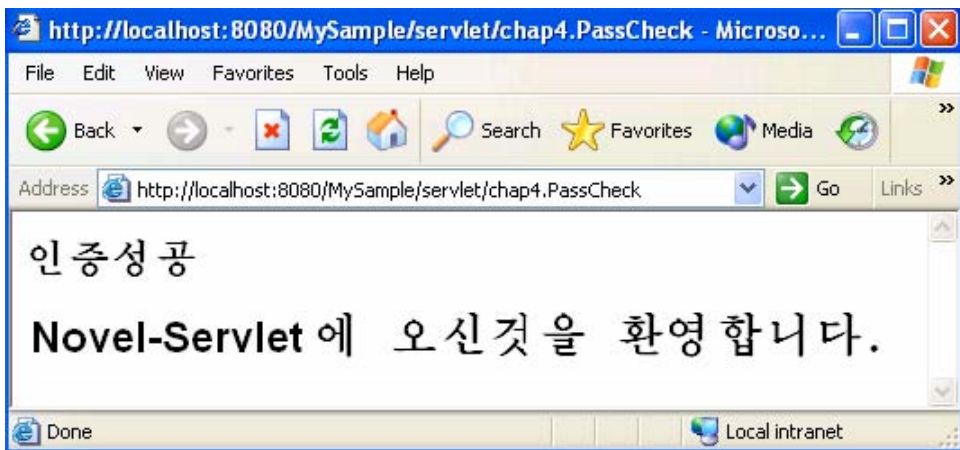



그림 4-20. 실레 4-15의 실행결과

프로그람설명

먼저 base64로 부호화된 문자렬을 복호화하기 위해 BASE64Decoder클라스를 적재한다. 즉

import sun.misc.BASE64Decoder;

우의 클라스는 JDK1.1이후 판본에 모두 포함되여있지만 공식적으로 쓰도록 허용되여있지 않기때문에 문서에는 구체적으로 설명되여있지 않다.

다음으로 머리부에서 Authorization을 받아서 그것이 null값이면 응답메쏘드로 상태와 머리부를 설정하여 인증을 위한 대화칸이 나온다. 결과를 보면 여기서 설정한 realm이 령역으로 표시되였다는것을 확인할수 있다.

String authorization = request.getHeader("Authorization");

```
if (authorization == null) {
  response.setStatus(response.SC_UNAUTHORIZED);
  response.setHeader("WWW-Authenticate", "BASIC realm=\"novel JSP-Servlet\"");
}
```

만일 Authorization머리부에 값이 있으면 그 값을 받아 BASE64Decoder를 사용하여 복호화한다. 앞에 붙은 basic란 단어를 빼기 위하여 substring을 리용하는데 아래와 같이 설정하여 6번째 글자부터 읽어 문자렬로 저장한다.

```
String userInfo = authorization.substring(6).trim();
BASE64Decoder decoder = new BASE64Decoder();
String info = new String(decoder.decodeBuffer(userInfo));
```

복호화하여 나온 결과가 《id:password》로 이루어져있으므로 id와 password를 구분하기 위하여 《:》을 index에 넣는다.

```
int index = info.indexOf(":");
String user = info.substring(0, index);
String password = info.substring(index+1);
```

여기서는 비교할 사용자식별자와 통과암호를 코드안에 써놓았다. 실제적으로는 자료기지나 다른 파일 등에서 불러와야 한다. 이것과 사용자가 입력한 인증정보와 비교하여 맞으면 원하는 페지에 접근할수 있고 틀리면 다시 머리부를 설정하는 부분으로 돌아가게 된다.

```
String realUser = "Admin";
String realPassword = "1234";
if (realUser.equals(user) && realPassword.equals(password)) {
```

다음으로 Http의 상태에 대하여 더 고찰한다. Http의 상태를 설정하면 보다 폭넓은 프로그람작성을 할수 있으며 상태코드를 분석하여 어떤 곳에서 오유가 발생하고 어떻게 결과가 처리되였는가를 알수 있다. 아래의 표에서는 상태코드와 그의 의미들을 서술한다.

표 4-13. 상태코드의 종류와 그 의미

100~199 의뢰기가 다음에 행동할 정보를 제공한다.	
100(continue)	의뢰기에서 추가문서(attatched document)를 보내기 전에 요청을 Expect머리부에 설정해서 보낸다.
101(switching protocol)	봉사기가 Upgrade머리부에 응답할수 있고 다른 규약으로 전환한다는것을 나타낸다.
200~299 요청받은 내용이 성공적으로 처리되였다는것을 나타낸다.	
200(OK)	모든것이 성공적으로 처리되였다는것을 나타낸다.
201(Created)	의뢰기의 요청에 대하여 봉사기가 새로운 문서를 생성하였다는것을 나타낸다.

202(Accepted)	의뢰기가 요청을 받아들이기만 했을뿐 아직 완료되지 않은 상태를 나타낸다.
204(No Content)	새로운 문서가 없어 열람기에게 이전 문서를 계속 표시하라고 알려준다.
205(Reset Content)	새로운 문서가 없더라도 열람기에서 창을 초기화하고 문서를 새로 표시한다.
300~399 파일들이 이동되였을 때 이동하는 위치를 나타낸다. Location머리부에 이에 대한 응답이 포함된다.	
300(Multiple Choice)	요청된 문서가 여러곳에 있을 때 어떤 문서를 원하는가를 문의한다.
301(Moved Permanently)	요청된 문서의 위치가 영구적으로 변하였다는것을 나타낸다.
302(Found)	요청된 문서의 위치가 림시적이라는것을 나타낸다.
304(Not Modified)	열람기의 고속완충기억기에 들어있는 문서가 최신 문서이므로 그것을 그대로 사용해도 된다는것을 나타낸다.
305(Use Proxy)	대리봉사기를 통해서 요청된 문서를 전송받아도 된다는것을 나타낸다.
400~499　의뢰기에 의한 오유를 나타낸다.	
401(Bad Request)	의뢰기에서 틀린 명령문으로 요청하였음을 나타낸다.
402(Unauthorized)	의뢰기가 Authorization머리부에 틀린 인증정보를 넣었음을 나타낸다.
403(Forbidden)	의뢰기의 인증정보에 상관없이 페지에 대한 접근을 거부한다는것을 나타낸다.
404(Not Found)	의뢰기가 요청한 자원이 봉사기에 없다는것을 나타낸다.
408(Request Timeout)	의뢰기의 요청을 접수하는 시간이 지났다는것을 나타낸다.
410(Conflict)	요청된 문서는 없어지고 새로운 주소는 알수 없다는것을 나타낸다.
413(Request Entity Too Large)	요청된 문서는 봉사기가 처리할수 있는 문서의 크기보다 크다는것을 나타낸다.
415(Unsupported Media Type)	요청된 문서는 봉사기가 처리할수 없는 문서라는것을 나타낸다.
500~599　봉사기에 의한 오유를 나타낸다.	
500(Internal Server Error)	봉사기내부에 문제가 있음을 나타낸다.
501(Not Implemented)	의뢰기의 요청을 처리하는데 필요한 기능을 봉사기가 지원하지 않음을 나타낸다.

502(Bad Gateway)	봉사기가 관문역할을 할 때 원격봉사기로부터 틀린 응답을 받았음을 나타낸다.
503(Service Unavailable)	봉사기의 사정으로 응답할수 없음을 나타낸다.

제8절. ServletContext클라스

4.8.1. ServletContext란

하나의 Servlet가 Servlet용기와 통신하기 위하여 사용하는 메쏘드들을 가지고있는 클라스가 바로 ServletContext이다. 그러나 ServletContext자체의 의미를 모르면 단순히 Servlet용기와 통신하여 Servlet의 정보를 얻는다는 결론밖에 얻지 못한다.

하나의 웨브응용프로그람안에는 하나의 문맥이 존재한다. 이미 웨브응용프로그람의 의미를 1장에서 학습하였다. 웨브의 프로그람적인 의미는 등록부의 개념으로서 Servlet용기안에서 응용프로그람단위로 Servlet를 관리한다는것이다.

웨브응용프로그람안에 있는 모든 Servlet들을 관리하고 정보를 공유할수 있는 역할을 담당하는것이 ServletContext이다. 이러한 내용을 아래에서 그림으로 보여주었다(그림 4-21).

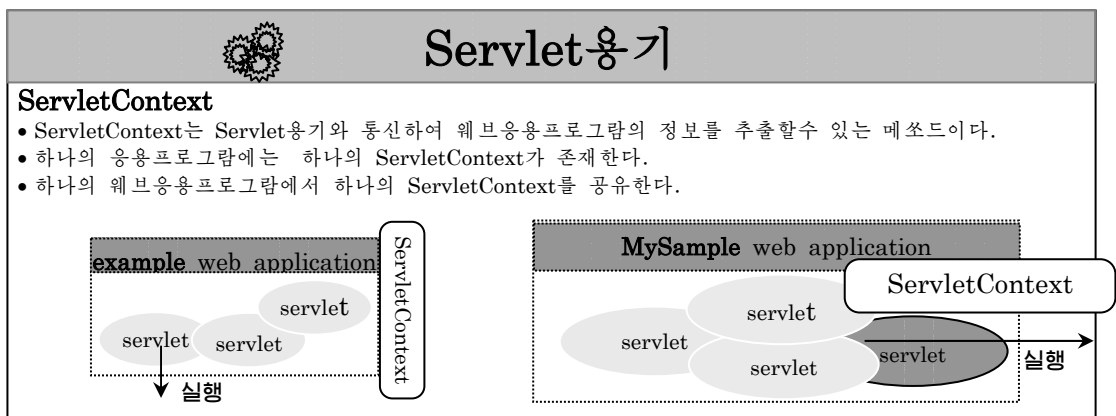



그림 4-21. Servlet용기

ServletContext는 Servlet의 정보를 추출하려는 경우 Servlet용기에 접근해야 하는데 이러한 접근을 가능하게 해준다. 이때 Servlet 하나하나에 접근하는것이 아니라 응용프로그람단위로 접근하게 한다.

- **ServletContext를 얻는 방법**

ServletContext는 ServletConfig의 getServletContext()메쏘드를 사용하여 얻는다. Servlet는 HttpServlet를 계승하고 HttpServlet는 Servlet Config를 실현하므로 getServletContext()메쏘드를 직접 리용할수 있다.

- ServletContext sc = getServletContext(); //Servlet안에서 사용가능

ServletContext는 Servlet정보를 추출하기 위하여 많이 사용한다. Servlet는 봉사기측프로쎄스의 범주내에서 생성되고 소멸된다. 이러한 Servlet에서는 초기화정보를 얻거나 실행환경에 대한 정보를 얻어낼수 있다. Servlet초기화정보는 Servlet가 시작되는 시기에 리용하며 실행환경에 대한 정보는 Servlet가 실행중인 경우에는 언제나 리용할수 있다. 아래에서는 이러한 정보들을 추출하는 메쏘드들을 설명한다.

- 파일의 MIME자료형을 얻는 메쏘드 → getMimeType()
- a log file에 기록하는 메쏘드→ log()
- Servlet자체의 정보를 얻어내는 메쏘드→ getServletInfo()
- Servlet의 가상등록부의 실제경로를 얻는 메쏘드→ getRealPath()
- Servlet의 판본확인메쏘드 → getMajorVersion(),getMinorVersion()

4.8.2. ServletContext의 활용

하나의 웨브응용프로그람안에 하나의 Servlet문맥이 존재한다는데 대하여 앞에서 설명하였다. 즉 같은 응용프로그람안에서는 Servlet문맥을 사용하여 정보를 공유할수 있고 Servlet들을 다룰수 있다.

아래에서 Servlet문맥을 리용한 Servlet들사이의 정보공유와 다른 웨브응용프로그람에서 현재의 웨브응용프로그람의 Servlet문맥정보를 사용할수 없다는 내용을 확인하는 실례를 보여주었다.

실례 4-16

SampleObject.java

```
package chap4;
import java.util.*;
public class SampleObject{
   private Timer timer=null;
   private Date startDate=null;
   private String intervalDate;
   private boolean isCont=true;
   public SampleObject(){
      this.startDate= new Date();
```

```
    }
    public void startTimeStamp(){
        timer=new Timer();
        timer.start();
    }
    public void endTimer(){
        this.isCont= false ;
    }
    public String getInterval(){
       return intervalDate;
    }
    class Timer extends Thread{
       public void run(){
          while(isCont){
             try{
                sleep(1000);
             }catch(InterruptedException ex){
                break;
             }
             long gap = new Date().getTime() - startDate.getTime();
             gap = gap/1000;
             intervalDate = (gap/60) +"분 "  + (gap%60) + "초 ";
          }
       }
    }
}//End of SampleObject
```

프로그람설명

이 프로그람은 실행된 때로부터 시간이 얼마나 지났는가를 계산하는 프로그람이다. 즉 startTimeStamp메쏘드를 호출하고 이에 관련된 Timer클라스의 토막처리를 실행시켜 시간을 측정할수 있다.

```
public void startTimeStamp(){
timer=new Timer();
timer.start();
... ...
public void run(){
while(isCont){
try{
```

```
sleep(1000);
}catch(InterruptedException ex){
break;
}
```

그리고 getInterval메쏘드를 호출하면 제일 처음에 호출된 때부터 다시 호출할 때까지의 시간간격을 계산하여 귀환한다.

```
public String getInterval(){
return intervalDate;
}
... ...
intervalDate = (gap/60) +"분 " + (gap%60) + "초 ";
```

다음의 프로그람은 Servlet를 초기화할 때 init의 참조파라메터로 되는 ServletConfig객체를 사용하여 ServletContext객체를 얻은 다음 ServletContext에 우에서 만든 시간계산프로그람의 객체를 저장해놓음으로써 다른 Servlet에서도 ServletContext를 사용하여 시간계산프로그람의 리용을 가능하게 한다.

실례 4-17

ContextObjectTest.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ContextObjectTest extends HttpServlet{
   public void init(ServletConfig cnf) throws ServletException{
      super.init(cnf);
      ServletContext cnt=cnf.getServletContext();
      SampleObject obj=new SampleObject();
      obj.startTimeStamp();
      cnt.setAttribute("obj",obj);
      System.out.println("ContextObjectServlet.init()");
   }
   public void doGet(HttpServletRequest req,HttpServletResponse res)
            throws IOException, ServletException{
      res.setContentType("text/html;charset=big5");
      PrintWriter out = res.getWriter();
      out.println("<HTML><BODY><CENTER>");
      ServletContext cnt=getServletContext();
      Object cobj=cnt.getAttribute("obj");
```

```
        if(cobj!=null){
           SampleObject obj=(SampleObject)cobj;
           out.println("<H1>" + obj.getInterval() + "</H1>");
        }
        else{
           out.println("SampleObject not found!");
        }
        out.println("</CENTER></BODY></HTML>");
     }
     public void destroy(){
        ServletContext cnt=getServletContext();
        SampleObject cobj=(SampleObject)cnt.getAttribute("obj");
         cobj.endTimer();
        cnt.removeAttribute("obj");
        System.out.println("ContextObjectServlet.destroy()");
     }
}
```

프로그람설명

먼저 ServletConfig객체를 사용하여 ServletContext객체를 얻은 다음 ServletContext에 우에서 만든 시간계산프로그람(실례 4-16)에 있는 객체를 《obj》라는 이름으로 지정한다.

```
ServletContext cnt=cnf.getServletContext();
SampleObject obj=new SampleObject();
obj.startTimeStamp();
cnt.setAttribute("obj",obj);
```

이렇게 하여 다른 Servlet에서도 obj객체를 사용할수 있게 되였다.

obj.startTimeStamp()을 리용하여 시간측정을 진행하고있다.

doGet메쏘드부분을 보면 ServletContext객체를 사용하여 우에서 지정한 obj를 강제형변환의 방법으로 얻어 우의 시간계산프로그람을 사용한다. 결과 시작후 시간이 얼마나 지났는가를 출력된다.

```
ServletContext cnt=getServletContext();
Object cobj=cnt.getAttribute("obj");
if(cobj!=null){
SampleObject obj=(SampleObject)cobj;
out.println("<H1>" + obj.getInterval() + "</H1>");
}
```

그림 4-22는 우의 실례의 실행결과를 보여준다.

그림 4-22. 실례 4-17의 실행결과

아래의 실례는 같은 웨브응용프로그람안에서 우에서 지정한 Servlet문맥정보를 리용하는 프로그람이다. 실행결과는 그림 4-23에서 보여주었다.

실례 4-18

ContextObjectTest2.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ContextObjectTest2 extends HttpServlet{
   public void doGet(HttpServletRequest req, HttpServletResponse res)
               throws IOException, ServletException{
      res.setContentType("text/html;charset=big5");
      PrintWriter out = res.getWriter();
      out.println("<HTML><BODY><CENTER>");
      ServletContext cnt=getServletContext();
      Object cobj=cnt.getAttribute("obj");
      if(cobj!=null){
         SampleObject obj=(SampleObject)cobj;
         out.println("<H1>" + obj.getInterval() + "</H1>");
      }
      else{
         out.println("SampleObject not found!");
      }
      out.println("</CENTER></BODY></HTML>");
   }
}
```

그림 4-23. 실례 4-18의 실행결과

프로그람설명

ServletContext객체를 생성하여 Servlet문맥이 가지고있는 obj속성을 얻은 다음 그것을 원래의 형인 SampleObject형으로 강제형변환하여 객체를 사용하고있다.

```
ServletContext cnt=getServletContext();
Object cobj=cnt.getAttribute("obj");
if(cobj!=null){
SampleObject obj=(SampleObject)cobj;
out.println("<H1>" + obj.getInterval() + "</H1>");
}
```

다음은 다른 웨브응용프로그람에서 우의 응용프로그람에 접근하여 Servlet문맥의 정보를 사용하는 경우의 실례를 보기로 하자.

실례 4-19

ContextObjectTest2.java

```
package chap4;
import chap4.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ContextObjectTest2 extends HttpServlet{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
                  throws IOException, ServletException{
      res.setContentType("text/html;charset=big5");
      PrintWriter out = res.getWriter();
      out.println("<HTML><BODY><CENTER>");
```

```
        ServletContext cnt=getServletContext();
        Object cobj=cnt.getAttribute("obj");
        if(cobj!=null){
            SampleObject obj=(SampleObject)cobj;
            out.println("<H1>" + obj.getInterval() + "</H1>");
        }
        else{
            out.println("SampleObject not found!");
        }
        out.println("</CENTER></BODY></HTML>");
    }
}
```

프로그람설명

이 프로그람은 실례 4-18의 ContextObjectTest2클라스를 다른 웨브응용프로그람에 넣어 실행한것이다. 즉 원래의 응용프로그람인 MySample등록부와는 다른 ContextSample이라는 등록부를 만들어 그 아래 classes등록부의 chap4등록부에 생성하여 넣었다. 결과 ServletContext에서 obj를 찾지 못하고 《SampleObject not found》라는 통보문이 현시되였다(그림 4-24).

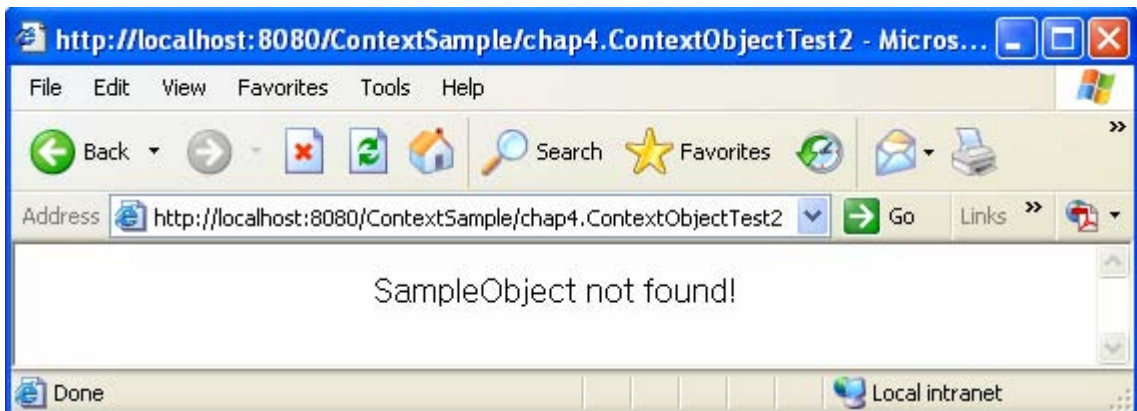



그림 4-24. 실례 4-19의 실행결과

제9절. RequestDespatcher대면

4.9.1. forward메쏘드

RequestDespatcher대면은 의뢰기로부터 요청받은 정보를 봉사기의 HTML, JSP 또는 Servlet 등의 자원에 보내는 역할을 하는 대면이다.

이 대면은 요청받은 내용을 Servlet에서 다른 Servlet, JSP 또는 HTML파일 등으로 보내주는 forward메쏘드와 다른 페지의 내용을 Servlet에 포함시켜 응답하는 include메쏘드로 구성된다. 즉

public void forward(ServletRequest request, ServletResponse response)

public void include(ServletRequest request, ServletResponse response)

여기에서는 forward메쏘드에 대하여 보고 다음 소절에서 include메쏘드에 대하여 설명한다.

forward메쏘드는 의뢰기의 요청 request와 응답 response를 다른 servlet에 넘겨주어 나머지 작업을 전부 그것에 맡긴다. 또한 다른 페지로 회송(forwarding)될 때 조종권한까지 회송된다(그림 4-25).

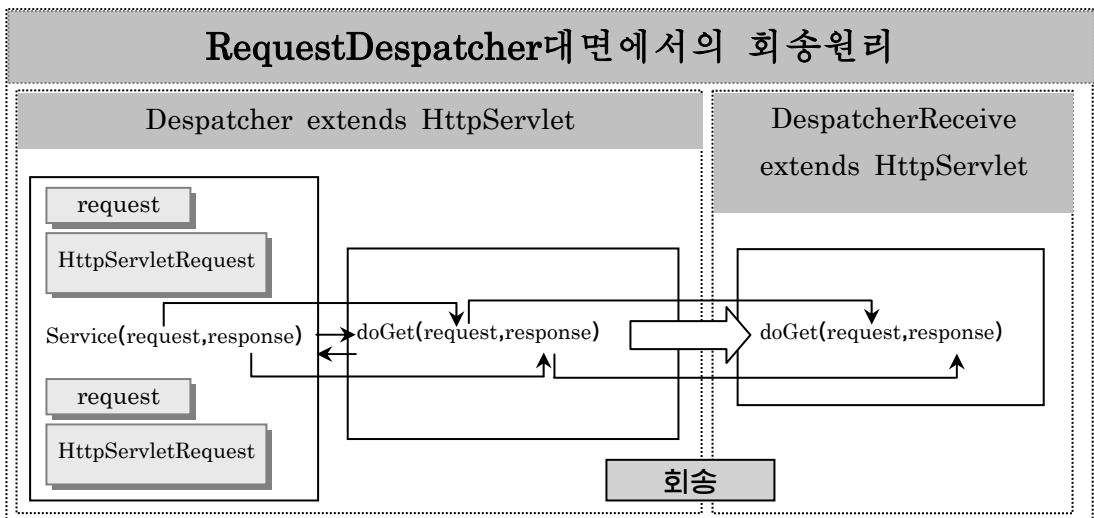



그림 4-25. RequestDespatcher의 회송원리

다음의 실례는 RequestDespatcher의 forward메쏘드를 사용하여 의뢰기로부터 받은 정보를 다른 Servlet로 보내여 출력하는 실례이다.

실례 4-20

DespatcherForward.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DespatcherForward extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException{
        ServletContext sc = getServletContext();
        RequestDispatcher rd = sc.getRequestDispatcher("/servlet/chap4.DespatcherReceive");
        rd.forward(request, response);
    }
}
```

DespatcherReceive.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DespatcherReceive extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException{
        response.setContentType("text/html;charset=big5");
        String message = request.getParameter("command");
        PrintWriter out = response.getWriter();
        out.println("<HTML><BODY>");
        out.println("DespatcherReceive Servlet");
        out.println("<H1>"+ message + "</H1>");
        out.println("</BODY></HTML>");
    }
}
```

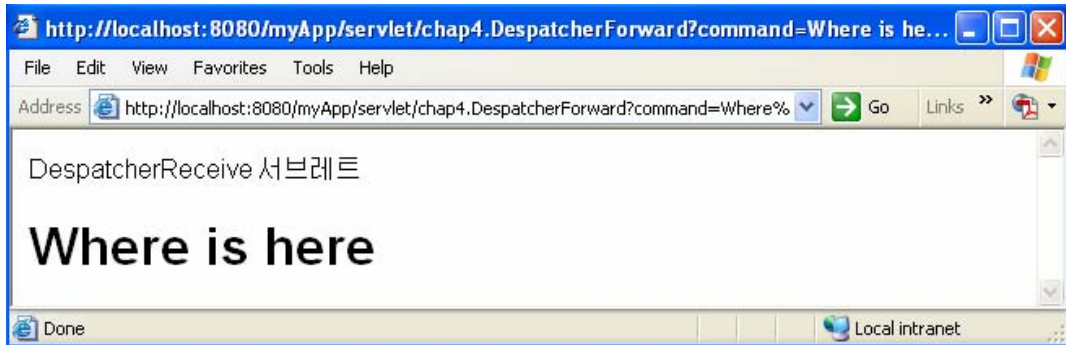



그림 4-26. 실례 4-20의 실행결과

프로그람설명

doGet메쏘드를 사용하여 의뢰기의 정보를 받아 회송하기 위하여 ServletContext를 얻는다. ServletContext는 getServletContext()메쏘드를 리용하여 얻을수 있다. 생성된 ServletContext객체를 리용하여 RequestDispatcher를 생성한다. 이때 회송할 Servlet의 주소를 입력하여 RequestDispatcher를 생성한다. 다음 RequestDespatcher의 forward메쏘드를 사용하여 DespatcherReceive로 사용자요청을 회송한다.

```
ServletContext sc = getServletContext();
RequestDispatcher rd =
        sc.getRequestDispatcher("/servlet/chap4.DespatcherReceive");
rd.forward(request, response);
```

DespatcherReceive.java에서는 사용자요청정보를 받아 HTML형식을 만들어 화면에 출력해준다.

```
String message = request.getParameter("command");
out.println("<H1>"+ message + "</H1>");
```

프로그람을 실행하기 위하여 우선 웨브열람기에서 통보문을 입력하면 RequestDespatcher Servlet가 호출된다. 그러면 그 정보가 DespatcherReceive Servlet로 넘겨져 출력된다는것을 알수 있다(그림 4-26).

여기서 한가지 흥미있는것은 웨브열람기의 주소칸을 보면 알수 있는것처럼 다른 Servlet로 사용자정보를 넘기면 정보를 받은 페지에서 요청을 처리하여 출력까지 해주지만 웨브열람기의 주소칸은 원래 호출한 그 주소가 바뀌지 않는다는것이다. 이러한 원리를 리용하면 사용자의 요청정보를 분석하여 그에 맞는 다른 페지로 요청을 보내여 처리해도 마치 하나의 페지에서 처리되는듯한 효과를 얻을수 있다.

4.9.2. include메쏘드

우에서는 RequestDespatcher의 forward메쏘드를 리용하여 사용자요청정보를 다른 Servlet로 보내는 원리에 대하여 고찰하였다. 여기서는 RequestDespatcher의 include메쏘드를 리용하여 다른 Servlet를 현재의 Servlet에 포함시키는 원리에 대하여 고찰한다.

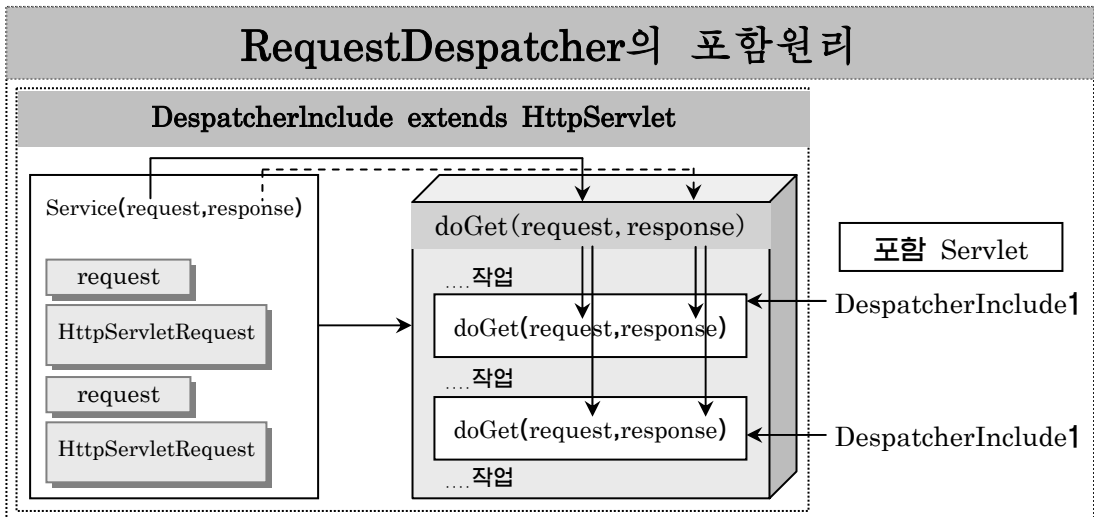



그림 4-27. RequestDespatcher의 포함원리

실례 4-21

DespatcherInclude1.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DespatcherInclude1 extends HttpServlet{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
               throws IOException, ServletException{
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      out.println("<H1> 부르셨습니까? </H1>");
      out.println("포함된 DespatcherInclude1써브레트");
      out.println("");
   }
}
```

DespatcherInclude.java

```
package chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DespatcherInclude extends HttpServlet{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws IOException, ServletException{
      response.setContentType("text/html;charset=euc-kr");
      PrintWriter out = response.getWriter();
      out.println("<HTML><BODY>");
      out.println("<H2> 안녕하십니까 </H2>");
      ServletContext sc = getServletContext();
      RequestDispatcher rd=sc.getRequestDispatcher("/servlet/chap4.DespatcherInclude1");
      rd.include(request, response);
      out.println("<H2> 다시  오십시오.. </H2>");
      rd.include(request, response);
      out.println("</BODY></HTML>");
   }
}
```

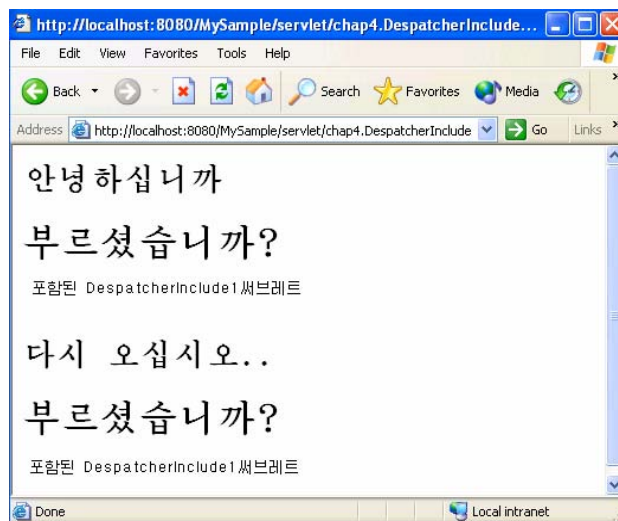



그림 4-28. 실례 4-21의 실행결과

프로그람설명

DespatcherInclude.java파일을 보면 다른 Servlet를 포함하기 위하여 RequestDespatcher의 include메쏘드가 리용되였다는것을 알수 있다. 여기서도 역시 RequestDespatcher의 객체를 얻기 위하여 ServletContext클라스를 사용하고있다.

```
ServletContext sc = getServletContext();
RequestDispatcher rd=sc.getRequestDispatcher
        ("/servlet/chap4.DespatcherInclude1");
```

여기서 흥미있는 점은 원천프로그람에서 보는것처럼 페지를 여러번 포함해도 된다는것이다.

```
rd.include(request, response);
```

이렇게 RequestDespatcher의 include메쏘드를 사용하면 Servlet뿐아니라 HTML, JSP 파일 등을 쉽게 포함하여 쓸수 있다. 이 메쏘드는 같은 내용이 여러 곳에서 필요한 경우 하나만 만들어놓고 여러 곳에서 포함하여 쓰는데 유용하다. 그림 4-26은 이 실례의 실행결과를 보여준다.

제10절. SingleThreadModel대면

Servlet는 다중스레드모형에 기초하고있다. 그러므로 Servlet는 의뢰기의 요청이 있을 때마다 요청을 처리한다. Servlet는 service()메쏘드를 동시에 호출하는것을 기본으로 하고있으므로 만일 service()메쏘드내부에 공유자원이 있을 때에는 문제가 발생하게 된다. 그러므로 동시에 service()메쏘드를 호출하지 못하도록 Servlet용기범위에서 방지하여야 한다. 이때 SingleThreadModel대면을 실현할수 있다.

SingleThreadModel대면은 어떤 메쏘드도 가지고있지 않으므로 Servlet뒤에 《implements SingleThreadModel》만 붙여주면 된다. 이와 같은 원리를 그림으로 나타내면 다음과 같다.

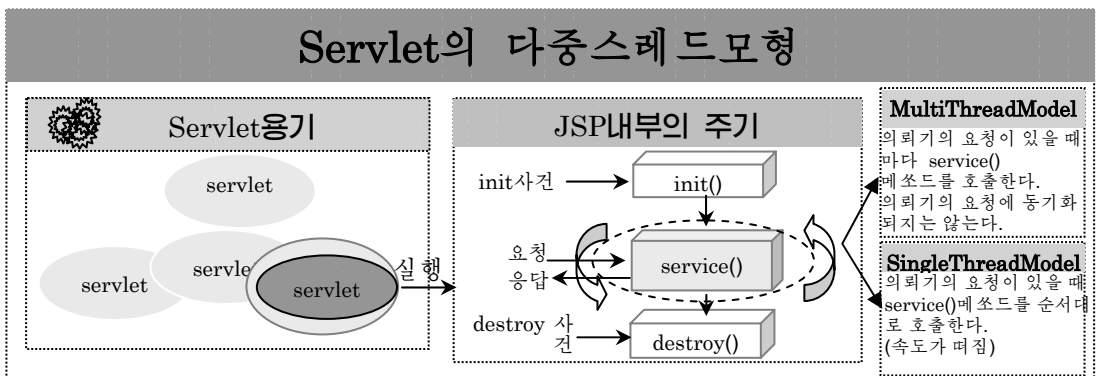



그림 4-29. Servlet의 다중스레드모형

SingleThreadModel은 요청을 순서대로 수행하므로 속도가 떠질수 있다. SingleThreadModel을 실현하더라도 동기화는 완전히 보장되지 않는다.

- SingleThreadModel을 사용하더라도 동기화를 보장하지 못하는 경우
  - 정적클라스성원변수를 쓰는 경우
  - Servlet령역외부에 있는 클라스를 리용하는 경우

이러한 경우에는 메쏘드에 synchronized열쇠어를 주어 해결할수 있다. 이러한 측면을 고려하면서 다음의 실례를 고찰해보자. 실행결과는 그림 4-30에서 보여준다.

실례 4-22

NotSingleThreadModelTest.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class NotSingleThreadModelTest extends HttpServlet implements Runnable {
   public int value = 5000;
   public void run(){
      try{
        Thread.sleep(2000);
      }catch(Exception e){}
      this.value = value-6000;
   }
   public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      out.println("<H1>NotSingleThreadModelTest</H1><BR>");
      out.print("최초 금액 : " +value+"<BR>");
      if(value>0){
         new Thread(this).start();
         try{
            Thread.sleep(3000);
         }catch(Exception e){}
         out.print("수정금액 : " +value+"<BR>");
         } else{
         out.println("잔액부족");
        }
        out.close();
    }
}
```

그림 4-30. 실례 4-22의 실행결과

프로그람설명

우의 실례는 잔액이 0보다 클 때 잔액에서 6000을 빼는 작업을 하는 토막처리가 동작하는 프로그람이다. 잔액이 뺄 금액보다 작아도 되지만 0보다는 커야 한다. 그런데 우의 결과화면에서 보는바와 같이 2개의 열람기창문이 현시된 상태에서 거의 동시에 접근을 하게 되면 먼저 실행된 토막처리가 잔액에서 6000을 빼기 전에 두번째 토막처리가 잔액을 비교하게 되는데 아직 첫번째 토막처리가 빼기를 하지 않았으므로 0보다 크다고 인식하고 토막처리를 발생하게 된다.

이러한 문제를 해결하기 위하여 나온것이 SingleThreadModel대면이다. 그러면 우의 실례에서 SingleThreadModel대면을 실현한 다음 프로그람을 우와 같은 방식으로 집행해보자.

실례 4-23

SingleThreadModelTest.java

```
package chap4;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SingleThreadModelTest extends HttpServlet
            implements Runnable, SingleThreadModel {
   public int value = 5000;
   public void run(){
```

```
      try{
         Thread.sleep(2000);
      }catch(Exception e){}
      this.value = value-6000;
   }
   public void service(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
      response.setContentType("text/html;charset=big5");
      PrintWriter out = response.getWriter();
      out.println("<H1>SingleThreadModelTest</H1><BR>");
      out.print("최초 금액 : " +value+"<BR>");
      if(value>0){
         new Thread(this).start();
         try{
            Thread.sleep(3000);
         }catch(Exception e){}
            out.print("수정 금액 : " +value+"<BR>");
      } else{
         out.println("잔액 부족");
      }
       out.close();
   }
}
```

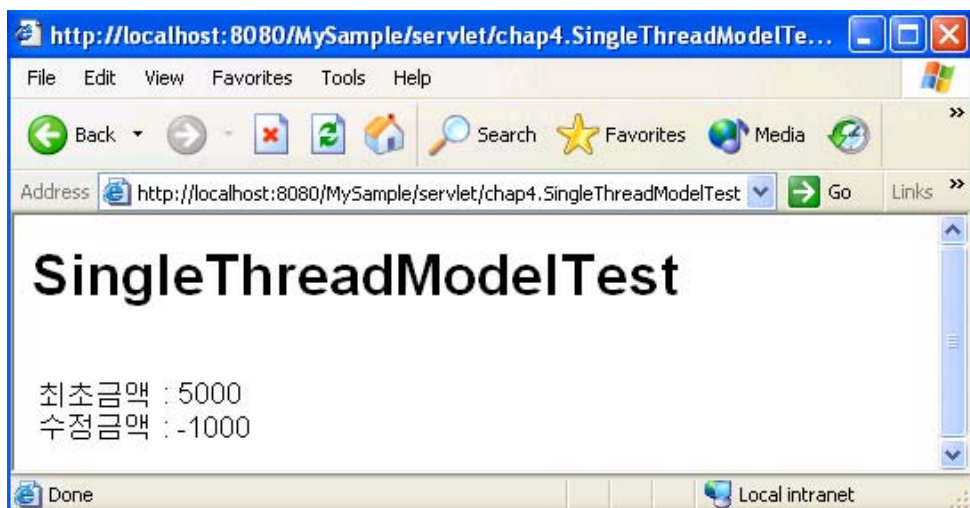
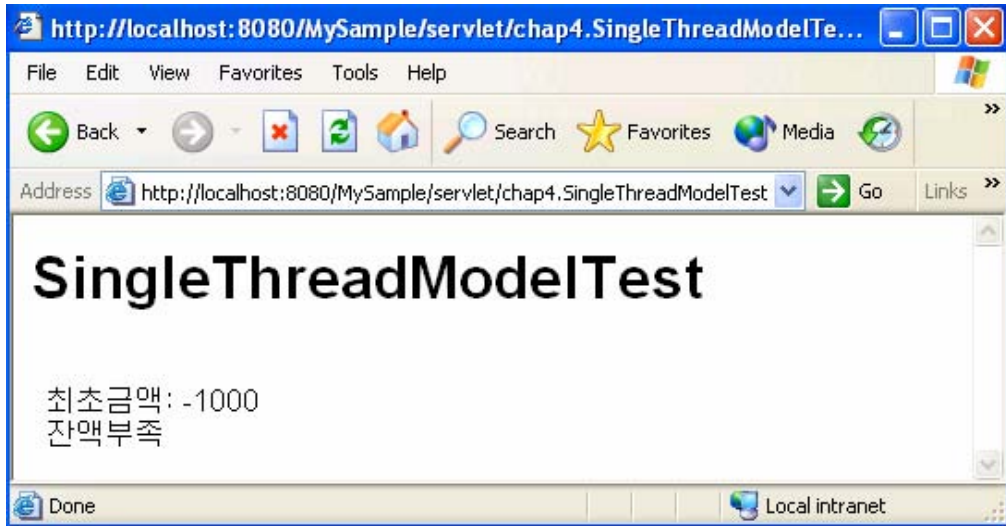



그림 4-31. 실례 4-23의 실행결과

프로그람설명

앞의 방식과 꼭같이 열람기창문을 2개 현시한 다음 동시에 접속하였어도 결과는 다르게 나온다. 이것은 SingleThreadModel대면이 토막처리를 하나하나 처리하였기때문이다. 그러나 앞에서 이야기한것처럼 SingleThreadModel대면만으로는 동기화를 조종할수 없다. 그림 4-31은 우의 실례의 실행결과를 보여준다.

제5장. JSP의 기본

제1절. JSP에서 사용하는 스크립트요소들

JSP는 Java언어를 기반으로 사용하는데 이것을 실현하자면 Java언어자체를 JSP에서 스크립트형태로 지원해주어야 한다. 이러한 스크립트형태의 프로그람을 지원하기 위하여 사용하는 꼬리표는 아래와 같다.

스크립트레트(Scriptlet) <% %>

표현식(Expressions) <%= %>

선언문(Declarations) <%! %>

지시문(Directives) <%@ %>

우의 4가지 꼬리표를 리용하여 JSP에서는 스크립트형식의 프로그람을 작성한다. 매개 꼬리표들의 역할을 아래에서 서술한다.

스크립트들과 그 역할

① 스크립트레트(Scriptlet) <% %> → _jspService에 프로그람을 추가
② 표현식(Expressions) <%= %> → _jspService에 프로그람을 추가 (out.print의 간략화)
③ 선언문(Declarations) <%! %> → .java파일에 성원마당과 성원메쏘드를 추가
④ 지시문(Directives) <%@ %> → .java파일에 여러가지 속성을 정하기(지시문의 형태에 따라 .java파일의 적당한 장소에 추가)

우의 스크립트요소들을 모두 리용하여 간단한 프로그람을 만들어보자.

실례 5-1

ScriptletTest.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<HTML><BODY>
<%!
   private  String title="주소록";
   private  String[] fName=new String[]{"이름","영철","인철","순철"};
   private  String[] fAddr=new String[]{"주소","평양","남포","개성"};
   public String jusoLine(int j){
   return fName[j]+" | "+fAddr[j];
 }
```

```
%>
<%= title %><p>
<%
 int a;
 for(a=0; a<fName.length; a++){
%>
<%=jusoLine(a)%><BR>
<% } %>
</BODY></HTML>
```

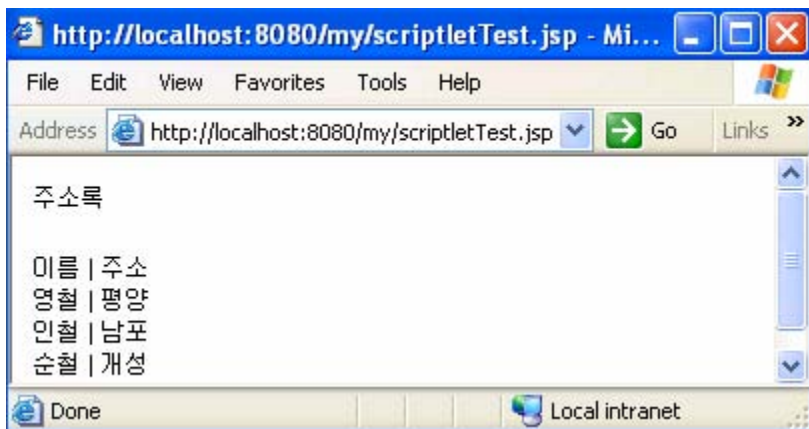



그림 5-1. 실례 5-1의 실행결과

프로그람설명

첫번째 행에서 사용한 <%@ %>형식의 명령문은 page지시문으로서 조선글부호화방식을 지정하기 위하여 리용한다. 지시문은 해당 페지에 여러가지 속성을 설정하는데 사용한다. 이 명령문은 contentType에서 웨브열람기로 text/html형식으로 전송한다는것과 charset에는 웨브열람기에서 받을 조선글의 부호화류형을 지정하고있다.

<%@ page contentType="text/html; charset=big5" %>

다음으로 <%! %>형식의 명령문은 페지전체에서 참조될 성원변수나 성원메쏘드를 선언할 때 사용하는 선언문이다. 여기서는 문자렬형의 비공개(private)변수인 title과 문자배렬형의 비공개변수인 fName, fAddr 그리고 문자배렬형변수의 첨수를 사용하여 한 행의 문자렬로 만들어주는 jusoLine()성원메쏘드를 생성할 목적으로 선언문을 사용하였다.

이렇게 성원변수나 성원메쏘드를 정의할 때 사용하는것이 선언문이다.

```
private String title="주소록";
private String[] fName=new String[]{"이름","영철","인철","순철"};
private String[] fAddr=new String[]{"주소","평양","남포","개성"};
```

```
public String jusoLine(int j){
  return fName[j]+"|"+fAddr[j];
}
```

그리고 <%= %>명령문은 HTML에 출력될 내용을 표현하기 위하여 사용하는 표현식이다. 표현식은 프로그람작성자가 out.print()명령문을 편리하게 사용하도록 표현한것이다. 따라서 표현식의 내부에는 반두점을 붙이지 말아야 한다. 우에서 사용된 표현식을 스크립트를 사용하여 표현하면 다음과 같다.

```
<%=title %> → <% out.print(title); %>
<%=jusoLine(a)%> → <% out.print(jusoLine(a)); %>
```

<% %>명령문은 스크립트레트로서 그 사이에 있는 모든 Java코드들은 그대로 Servlet로 변환된다. 스크립트레트에서 열린 코드블로크는 언젠가는 반드시 닫아주어야 한다. 그리고 HTML꼬리표인
도 for문안에서 하나의 요소로서 쓰일수 있다.

```
<%
int a;
for(a=0; a<fName.length; a++){%>
<%=jusoLine(a)%><BR>
<% } %>
```

실행결과는 그림 5-1에서 보여준다.

제2절. _jspService의 지역변수와 내장객체들

앞에서 _jspService메쏘드를 학습하면서 내장객체에 대하여 간단히 언급하였다. 이 절에서는 내장객체에 대하여 구체적으로 서술한다. 우선 _jspService메쏘드안의 구성요소부터 고찰한다.

_jspService의 구성

1. 파라메터2개
   ① HttpServletRequest request
   ② HttpServletResponse response
2. 필요한 지역변수
   ① PageContext pageContext = null;
   ② HttpSession session = null;
   ③ ServletContext application = null;
   ④ ServletConfig config = null;
   ⑤ jspWriter out = null;
   ⑥ Object page = this;
   ⑦ jspFactory _jspxFactory = null;
   ⑧ String _value = null;

```
3. 지역변수초기화
   ① _jspxFactory = jspFactory.getDefaultFactory();
   ② pageContext = _jspxFactory.getPageContext(this, request, response, "",
      true, 8192, true);
   ③ application = pageContext.getServletContext();
   ④ config = pageContext.getServletConfig();
   ⑤ session = pageContext.getSession();
   ⑥ out = pageContext.getOut();
```

이것들은 _jspService메쏘드의 지역변수들이지만 이 지역변수들은 아주 째이게 만들어져 초기화된다. 기본적으로 _jspService의 파라메터로 request와 response가 넘어오게 된다. 이것은 Servlet에서와 같은 원리로 넘어오게 된다. Servlet에서 request와 response는 service메쏘드의 파라메터로 넘어오게 되며 이것은 다시 의뢰기의 요청방식에 따라서 doGet와 doPost의 파라메터로 넘어가게 된다. 이러한 원리로 _jspService의 파라메터로 Servlet용기에서 request와 response를 넘겨주게 되는것이다.

그리고 제일 먼저 초기화되는 변수는 _jspxFactory라는 지역변수이다.

_jspxFactory객체는 jspFactory클라스의 정적메쏘드인 getDefaultFactory를 리용하여 생성한다. 이렇게 생성된 _jspxFactory객체를 사용하여 다른 내장객체를 생성하는 pageContext(페지문맥)를 만든다. 이 pageContext를 사용하여 다른 내장객체들을 생성하고 사용한다. 결과적으로 _jspxFactory객체가 다른 내장객체들을 생성하는것으로 된다. 그러나 jspFactory의 객체인 _jspxFactory는 JSP작성자가 직접 사용하지 않고 내부적으로 사용되며 JSP가 Servlet로 변환될 때 자동적으로 실행되는 객체이다. 이러한 내용을 그림 5-2에서 자세하게 보여주었다.

그림 5-2. _JspService에서 사용되는 내장객체의 구성

<% %>꼬리표에 있는 모든 프로그람요소들은 스크립트형식으로 해석되여 _jspService 메쏘드의 지역변수가 초기화된 아래부분에 삽입되므로 <% %>꼬리표안에서 마음대로 내장객체(request, response, pageContext, application, config, session, out)들을 사용할 수 있다. 동작원리상의 문제이지만 쉽게 말하면 Servlet에서 자주 사용되는 객체들을 미리 만들어 놓는다고 볼수 있다.

의뢰기에 자료를 전송하기 위하여서는 jspWriter out객체를 얻어내야 한다.

또한 초기화파라메터를 얻기 위하여 Servlet로부터 ServletConfig객체인 config를 얻어내야 한다. 그리고 Servlet용기와 통신하기 위하여서는 ServletContext객체인 application을 사용하여야 한다. 대화접속을 관리하기 위하여서는 HttpSession객체인 session객체를 확보하여야 작업을 할수 있다. 이러한 작업들을 JSP에서는 pageContext로서 전부 해결한다. 파라메터로 넘어오는 HttpServletRequest request와 HttpServletResponse response는 원리적으로 Servlet와 같은 방식으로 얻을수 있다.

Servlet에서 service()메쏘드의 파라메터와 JSP에서 _jspService()메쏘드의 파라메터는 동일한 방식으로 JSP와 Servlet에 전달된다.

Servlet와 JSP의 봉사되는 메쏘드들은 다음과 같다(표 5-1).

표 5-1. Servlet와 JSP의 봉사되는 메쏘드들

Servlet
service(ServletRequest request, ServletResponse response) doGet(HttpServletRequest request, HttpServletResponse response) doPost(HttpServletRequest request, HttpServletResponset response)
JSP
_jspService(HttpServletRequest request, HttpServletResponse response)

제3절. 내장객체 request, response

내장객체는 <% %>안에서 사용할수 있도록 자동적으로 생성되는 객체이다.

먼저 _jspService메쏘드의 파라메터로 되는 내장객체 request, response에 대하여 고찰한다. 이미 Servlet에서 이 객체들에 대하여 학습하였다. 그리고 Servlet와 JSP에서 파라메터로 사용되는 request와 response는 같다는것을 설명하였다.

아래의 그림 5-3은 의뢰기가 통보문을 요청할 때 HTTP봉사기의 처리과정을 나타내고있다.

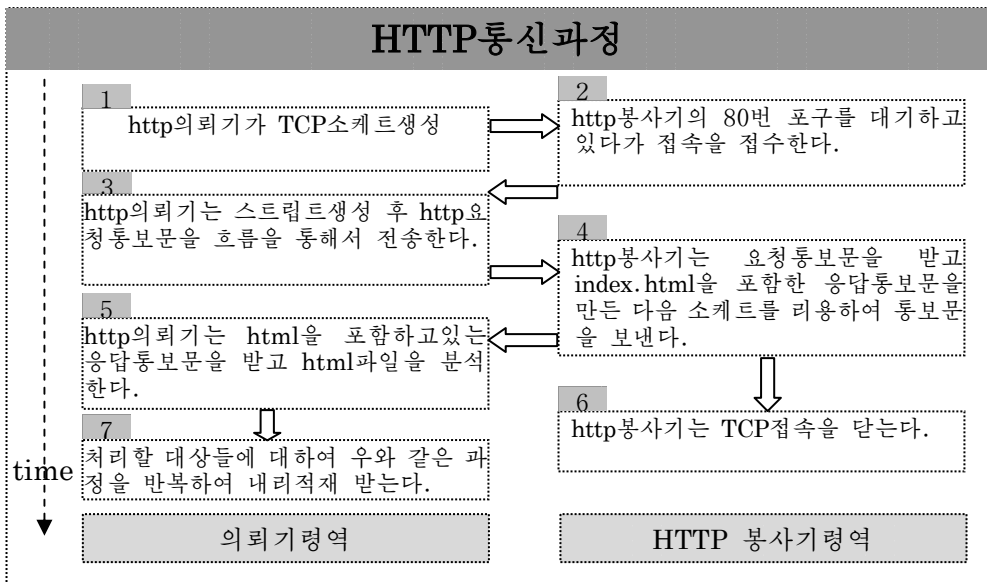



그림 5-3. HTTP에서 통신과정

먼저 요청통보문을 만들어야 한다. 그리고 요청통보문을 HTTP웨브봉사기로 보내면 응답자료를 받을수 있다. 이러한 과정을 JSP프로그람을 통해 보기로 하자. 실행결과는 그림 5-4에서 보여주었다.

실례 5-2

RequestMessage.jsp

```
<%@ page contentType = "text/html;charset=big5"%>
<%@ page import = "java.net.*,java.io.*"%>
<%
    InetAddress webServer = InetAddress.getByName("localhost");
    Socket httpPipe = new Socket(webServer, 80);
```

```
                                            //웨브봉사기포구:80
    InputStream is = httpPipe.getInputStream();
    PrintStream os = new PrintStream(httpPipe.getOutputStream());
    os.println("GET " + "/index.html" + " HTTP/1.0\n");
                                            //GET HTTP요청보내기
    int temp;
    CharArrayWriter caw= new CharArrayWriter();
    while ((temp = is.read())!=-1) {
      caw.write(temp);
    }
    out.write(caw.toCharArray());
    os.close();
    is.close();
%>
```

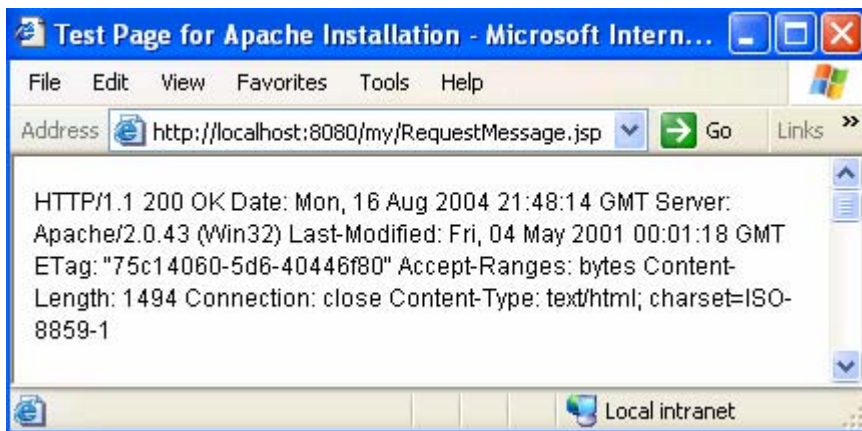



그림 5-4. 실례 5-2의 실행결과

프로그람설명

이 실례에서는 먼저 통신하기 위한 소케트를 생성하고있다. 현재 련결하려고하는 봉사기는 《http://localhost/index.html》이다. 목표 URL을 리용하여 소케트를 생성하는 방법은 다음과 같다.

```
InetAddress webServer = InetAddress.getByName("localhost");
Socket httpPipe = new Socket(webServer, 80);//웨브봉사기포구:80
```

소케트가 성공적으로 생성되면 해당 소케트에서 입력흐름과 출력흐름을 생성하여 웨브봉사기로 자료를 보내거나 자료를 받아들일수 있다. 다음의 명령문으로 생성된 소케트의 흐름을 얻고있다.

```
InputStream is = httpPipe.getInputStream();
```

```
PrintStream os = new PrintStream(httpPipe.getOutputStream());
```

흐름이 생성되면 출력흐름을 리용하여 련결된 웨브봉사기에 요청통보문을 보낸다.

```
os.println("GET " + "/index.html" + " HTTP/1.0\n");
```

출력흐름으로 요청통보문을 보내면 소케트에 련결된 입력흐름으로 자료가 들어온다. 자료를 CharArrayWriter에 적재하고 적재된 바이트들을 JSP의 출력흐름을 통해 보내고있다.

```
int temp;
CharArrayWriter caw = new CharArrayWriter ();
while ( (temp = is.read())!=-1) {
caw.write(temp);
}
out.write(caw.toCharArray());
```

여기서 Servlet와 차이나는 점은 Servlet의 실례에서는 ByteArrayStream을 사용하고 있지만 JSP에서는 CharArrayWriter를 사용하고있다는것이다.

제4절. 내장객체 pageContext

pageContext내장객체는 PageContext클라스형의 객체이다. 이 PageContext클라스는 다른 내장객체들을 생성하는 메쏘드를 가지고있는 추상클라스이다. 그러므로 이 클라스의 객체인 pageContext는 request, response를 제외한 내장객체중에서 제일 먼저 생성되는 객체이다. pageContext는 다른 내장객체를 생성할뿐아니라 JSP페지안의 조종권한을 정의하는 기능도 가지고있다.

5.4.1. pageContext의 생성과정

먼저 pageContext객체가 생성되는 과정을 고찰한다.

• pageContext객체를 생성하려면 우선 앞에서 설명한 _jspxFactory객체를 사용하여야 한다. 즉

pageContext=_jspxFactory.getPageContext(this,request,response, “”,true,8192, true);

pageContext는 _jspxFactory클라스의 getPageContext메쏘드를 리용하여 얻는다.

getPageContext의 형식은 다음과 같다.

PageContext getPageContext(Servlet servlet, ServletRequest request, ServletResponse response, String errorPageURL, boolean needsSession, int buffer, boolean autoflush)

아래의 실례를 통해 구체적으로 고찰해보자.

getPageContext(this,request,response, “”,true,8192, true);

여기서

this → 요청하는 Servlet를 지정(여기서는 자기자신이 요구하므로 this를 사용했음)

request, response → _jspService의 파라메터로 들어온것
"" → 오유가 생겼을 경우 현시할 오유페지의 URL주소를 입력하는 곳
true → JSP가 대화접속을 사용할것인가 물어보는 곳
8192 → 의뢰기로 전송할 자료의 완충기(buffer)의 크기를 지정하는 곳
true → 의뢰기로 전송할 자료를 위하여 기억기에서 자동지우기를 하겠는가를 지정

• pageContext의 객체가 생성되면 이 객체를 사용하여 다른 내장객체를 생성한다. PageContext를 리용하여 생성할수 있는 객체는 다음과 같다.

```
ServletContext application = pageContext.getServletContext();
ServletConfig config = pageContext.getServletConfig();
HttpSession session = pageContext.getSession();
jspWriter out = pageContext.getOut();
```

5.4.2. pageContext의 실제의미

하나의 페지당 하나의 pageContext가 존재한다. 다시 말하여 JSP를 실행시켜 생기는 Servlet 하나당 하나의 pageContext가 존재한다. 이러한 pageContext는 해당 페지의 다양한 정보와 특성들을 관리하며 페지범위안에서 정보공유를 돕는 역할을 한다. 이렇게 정보공유의 의미에서 보면 PageContext클라스의 내장객체인 pageContext를 리용한 정보의 공유범위는 현재 페지로 제한된다는것을 알수 있다.

앞장에서 ServletContext에 대하여 학습하였다. 하나의 웨브응용프로그람에 하나의 ServletContext가 존재하므로 같은 응용프로그람안에서는 ServletContext를 리용하여 정보를 공유할수 있다. 이렇게 정보공유의 의미에서 ServletContext와 pageContext에 대해 고찰하면 그림 5-5와 같다.

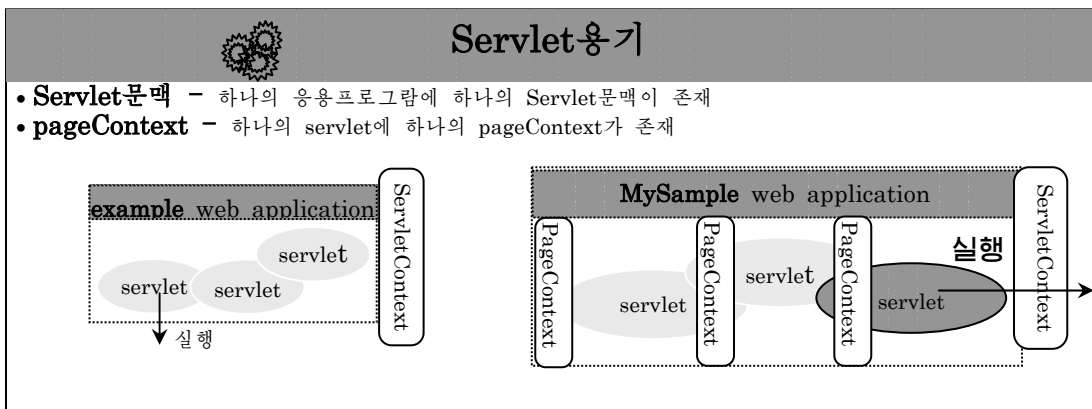



그림 5-5. Servlet용기

보다싶이 하나의 Servlet문맥으로 관리되는 여러개의 servlet는 자기 정보를 관리하는 pageContext를 각각 가지고있다. pageContext의 범위는 자기의 Servlet내부로 제한되여있다.

Servlet문맥과 pageContext의 범위	
Servlet문맥	하나의 응용프로그람에 하나의 Servlet문맥이 존재
pageContext	하나의 Servlet에 하나의 pageContext가 존재

pageContext를 리용한 정보의 공유가 현재의 페지안에서만 이루어지는가를 실례를 통하여 보자. 하나의 문자렬형의 변수값을 pageContext의 인수로 저장한 다음 그 페지안에서 pageContext를 리용하여 변수값을 여러번 읽어 공유가 되는가를 확인하고 또 다른 페지에서 pageContext를 통하여 이 변수를 공유하여 읽는것이 가능한가를 고찰한다.

실례 5-3

realPageContext.jsp

```
<%@ page contentType= "text/html;charset = BIG5"%>
<%
String real  = "realPageContext page";
pageContext.setAttribute("pageValue",real);
%>
<HTML><BODY>
<h2>real PageContext</h2>
<h3>** 페지문맥의 페지내 공유** <h3>
<h4>-호출-</h4>
<%
   Object cobj = pageContext.getAttribute("pageValue");
   if(cobj != null){
       String pValue = (String)cobj;
       out.println("<h3>page Value = " + pValue + "</h3>");

   }
   else{
       out.println("Attribute Not found!!");
   }
%>
<h4>-다시 호출-</h4>
<%
   Object dobj = pageContext.getAttribute("pageValue");
       String qValue = (String)dobj;
       out.println("<h3>page Value again =" + qValue + "</h3>");
%>
</BODY></HTML>
```

그림 5-6. 실례 5-3의 실행결과

프로그람설명

우선 문자렬형변수 real에 문자렬을 넣는다.

String real = "realPageContext page";

그것을 pageContext에서 pageValue란 이름으로 지정하고있다.

pageContext.setAttribute("pageValue",real);

다시 pageContext에서 《pageValue》란 이름으로 지정되여있는 값을 Object형으로 얻어낸다. 그 값을 검사하여 null값이 아니면 문자렬형으로 강제형전환하여 그 값을 출력한다. 만일 null값이면 《Attribute Not found》라는 통보문을 출력한다.

```
Object cobj = pageContext.getAttribute("pageValue");
if(cobj != null){
String pValue = (String)cobj;
out.println("<h3>page Value = " + pValue + "</h3>");
}
else{
out.println("Attribute Not found!!");
}
```

공유가 되는가를 확인하기 위하여 다시 한번 pageContext에서 《pageValue》값을 호출한다.

```
Object dobj = pageContext.getAttribute("pageValue");
String qValue = (String)dobj;
```

out.println("<h3>page Value again =" + qValue + "</h3>");

실행결과(그림 5-6)를 보면 pageContext로 지정한 변수값을 스크립트레트의 어디에서나 불러 쓸수 있다는것을 알수 있다.

아래의 실례는 우에서 지정한 변수값을 다른 페지에서 공유할수 있는가를 검사하는 실례이다.

실례 5-4

realPageContext1.jsp

```
<%@ page contentType= "text/html;charset = BIG5"%>
<HTML><BODY>
<h2>real PageContext</h2>
<h3>** 페지문맥의 페지내 공유** <h3>
<h4>-호출-</h4>
<%
  Object cobj = pageContext.getAttribute("pageValue");
  if(cobj != null){
        String pValue = (String)cobj;
        out.println("<h3>page Value = " + pValue + "</h3>");
  }
  else{
        out.println("Attribute Not found!!");
  }
%>
</BODY></HTML>
```

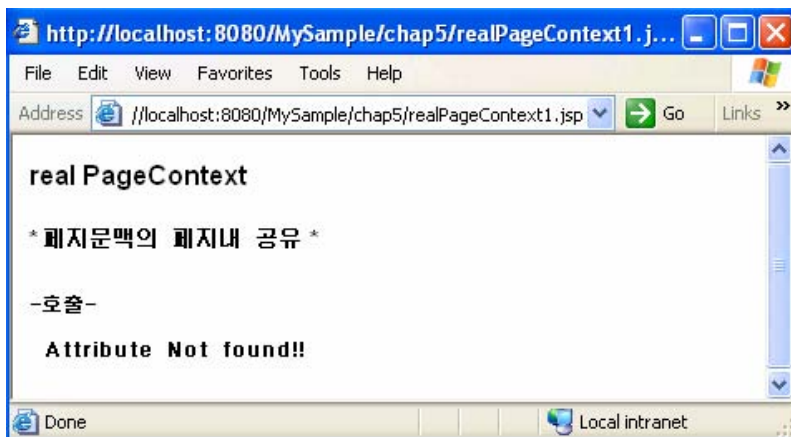



그림 5-7. 실례 5-4의 실행결과

프로그램설명

우와 마찬가지로 pageContext객체에서 《pageValue》라는 변수값을 찾아내여 출력하고있다. 그러나 결과(그림 5-7)에서 알수 있는것처럼 여기에서는 《pageValue》란 인수를 찾지 못하고 《Attribute Not found》라는 통보문만을 출력하고있다.

```
Object cobj = pageContext.getAttribute("pageValue");
if(cobj != null){
String pValue = (String)cobj;
out.println("<h3>page Value = " + pValue + "</h3>");
}
else{
out.println("Attrubute Not found!!");
```

이렇게 pageContext는 정보를 자기의 페지안에서만 공유할 때 사용한다.

5.4.3. forward, include

• pageContext의 forward(회송)는 현재 페지에서의 요청과 응답의 처리권한을 해당 URL로 완전히 넘기려고 할 때 리용한다. 그 결과 forward를 호출한 원래 페지의 요청과 응답은 전부 무시되고 회송된 페지의 요청과 응답을 처리하게 된다.

```
pageContext.forward( "해당URL" );
```

즉 회송요청이 일어나면 out.clearBuffer()를 호출하여 현재 페지의 완충기에 들어있는 내용을 모두 지워버린 다음 회송된 페지로 요청이 넘어간다. 그리고 회송된 페지에서는 출력완충기의 내용을 새로 담아서 응답을 다시 만들어 출력한다. 이것을 그림으로 나타내면 다음과 같다.

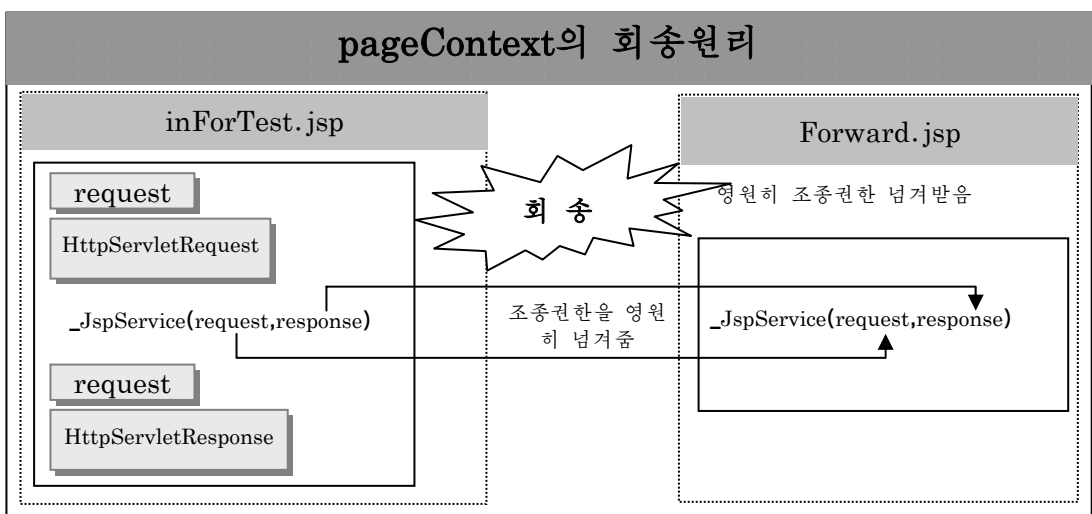



그림 5-8. PageContext의 회송원리

• 반면에 include는 현재 페지에서의 요청과 응답을 처리할 조종권한을 해당 URL로 잠시 넘겼다가 처리가 끝나면 다시 조종권한을 돌려받는다. 따라서 include를 호출하면 해당 URL로 요청과 응답 조종권한을 넘겨 처리하고 처리가 끝나면 그 결과와 함께 다시 원래의 페지에 조종권한을 돌려주어 그 페지의 처리를 끝낸다. 특정페지를 포함하는 명령문은 다음과 같다.

pageContext.include("해당URL");

include요청이 일어나면 그 페지에서는 요청하기전까지 해오던 작업들이 들어있는 완충기는 지워져 요청이 전달되는 페지로 넘어가며 해당 페지에서 림시로 조종권한을 가지고 작업을 하다가 작업이 끝나면 조종권한과 작업결과를 다시 원래의 페지로 모두 돌려준다.(그림 5-9)

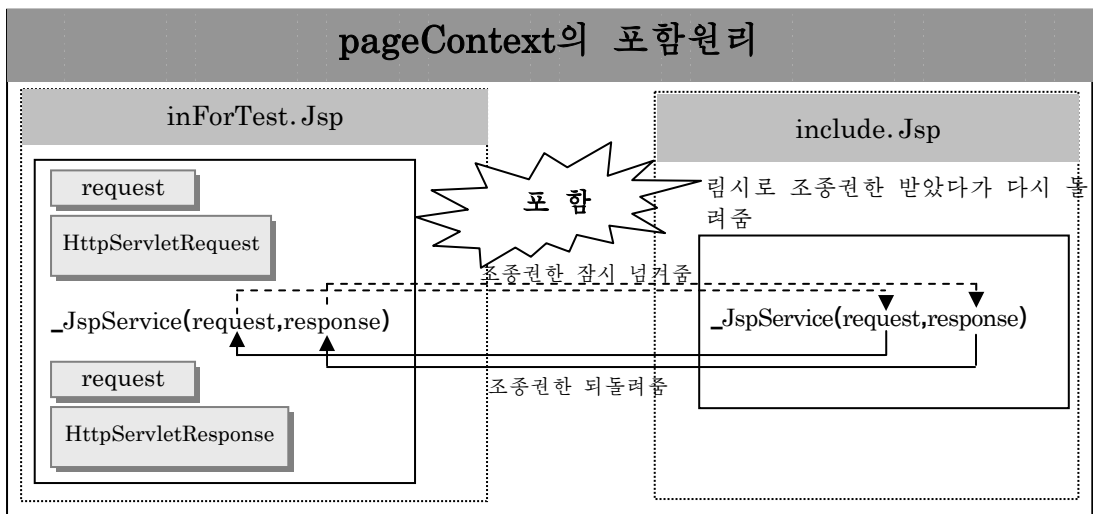



그림 5-9. PageContext의 포함원리

• 다음의 실례는 우의 그림에서 본 원리를 forward와 include를 리용하여 실현하는 실례이다. 이 실례의 실행결과를 그림 5-10에서 보여주었다.

실례 5-5

inForTest.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<HTML><BODY>
안녕하십니까.
<%
        if(request.getParameter("id").equals("jForward")){
                pageContext.forward("forward.jsp");
        }else if(request.getParameter("id").equals("jInclude")){
                pageContext.include("include.jsp");
```

```
        }else{
%>
        <%=request.getParameter("id") %>동무,<BR>
        forward와 include를 검증하는 페지입니다. <BR>
        getParameter의 id를 설정하십시오. <BR>
        forward를 검증하려면 jForward를,<BR>
        include를 검증하려면 jInclude를 입력하십시오.
<% } %>
</BODY></HTML>
```

forward.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<HTML><BODY>
반갑습니다. <%=request.getParameter("id") %>동무,<BR>
forward가 실행되였습니다.
</BODY></HTML>
```

include.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<h3><%=request.getParameter("id") %>동무,<BR>
include가 실행되였습니다.</h3>
```

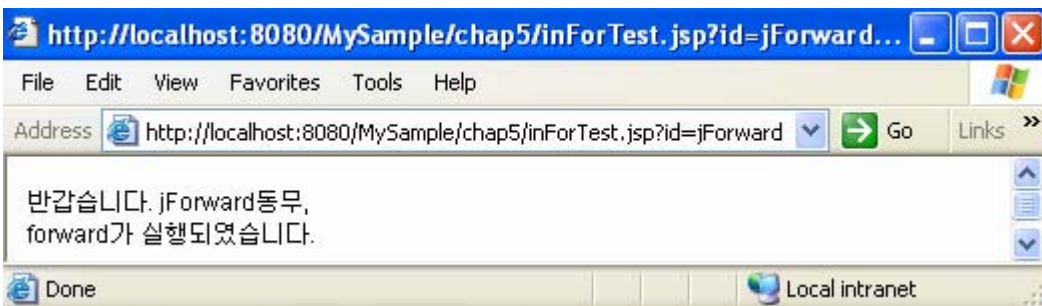
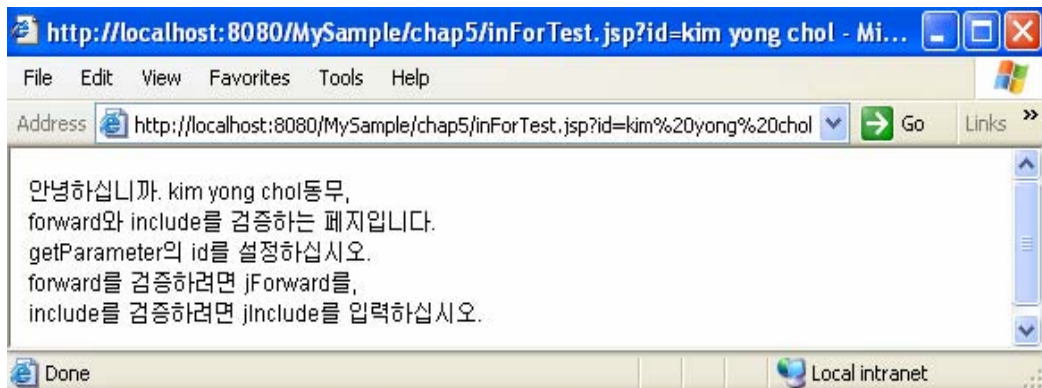
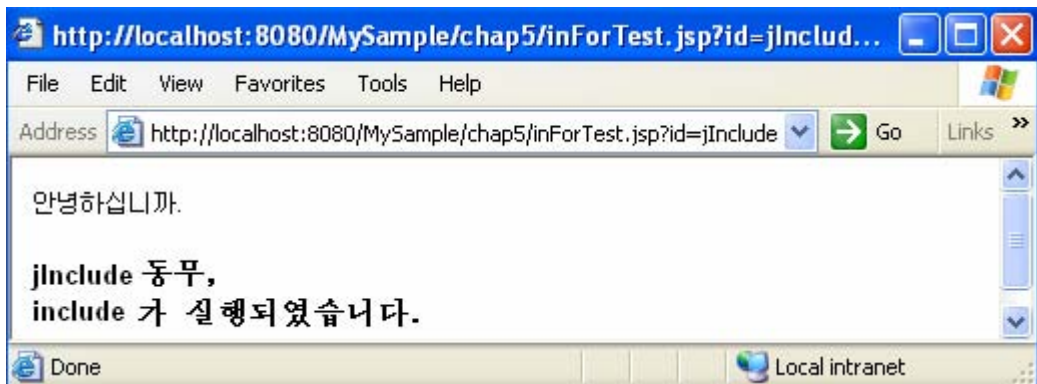



그림 5-10. 실례 5-5의 실행결과

프로그람설명

우의 실례는 3개의 JSP페지로 구성되여있다. inForTest.jsp에는 조건에 따라 forward.jsp로 회송시켜주는 forward()메쏘드와 include.jsp를 포함시켜주는 include() 메쏘드가 실현되여있다.

먼저 inForTest.jsp를 고찰하자. getParameter로 전달된 id값이 jForward라면 forward.jsp페지가 실행된다. 그런데 forward가 실행되기 전에 보여줄 《안녕하십니까.》라는 문장이 보이지 않는다. 그 리유는 forward()가 호출되면 자동적으로 지금까지 완충기에 저장된 내용을 지워주는 out.clearBuffer()가 실행되기때문이다.(그림 5-10에서 두번째 그림)

if(request.getParameter("id").equals("jForward")){

pageContext.forward("forward.jsp"); //현재의 페지는 지워진 다음 회송된다.

getParameter로 전달된 id값이 jInclude라면 include.jsp가 inForTest.jsp에 포함된다.

else if(request.getParameter("id").equals("jInclude")){

pageContext.include("include.jsp");

include.jsp는 inForTest.jsp에 삽입은 되지만 inForTest.jsp에 의하여 조종되지는 않는다. 따라서 HTTP전송양식을 새로 설정해주어야 한다.

<%@ page contentType="text/html; charset=big5" %>

<h3><%=request.getParameter("id") %>동무,

include가 실행되였습니다.</h3>

include.jsp의 처리가 끝나면 그 결과와 함께 조종권한을 다시 inForTest.jsp로 돌려준다.

입력된 id가 우의 두가지 경우와 맞지 않으면 else문이 실행되고 첫번째 그림이 출력된다.

<%=request.getParameter("id") %>동무,

forward와 include를 검증하는 페지입니다.

getParameter의 id를 설정하십시오.

forward를 검증하려면 jForward를,
include를 검증하려면 jInclude를 입력하십시오.
다음은 PageContext의 성원마당과 메쏘드들에 대하여 간단히 소개한다.(표 5-2, 5-3)

표 5-2. PageContext의 성원마당들

PageContext의 성원마당	
public static java.lang.String APPLICATION	PageContext이름표에 ServletContext를 저장하기 위해 사용하는 마당이다.
public static int APPLICATION_SCOPE	응용프로그람령역이라는 참조마당은 그것이 다시 리용될때까지 ServletContext에서 유효하게 된다.
public static java.lang.String CONFIG	PageContext이름표에 ServletConfig를 저장하기 위해 사용하는 마당이다.
public static java.lang.String EXCEPTION	이 마당은 ServletRequest속성목록과 PageContext이름표에 포착되지 않은 례외를 저장하는데 리용된다.
public static java.lang.String OUT	이 마당은 PageContext이름표에 현재의 jspWriter를 저장하는데 리용된다.
public static java.lang.String PAGE	이 마당은 PageContext이름표에 Servlet을 저장하는데 리용된다.
public static int PAGE_SCOPE	페지범위 - (이것은 기정이다) 이 참조마당은 현재의 Servlet.service로부터 귀환될 때까지 이 PageContext에서 유효하게 된다.
public static java.lang.String PAGECONTEXT	이 마당은 PageContext이름표에서 그것에 저장된 pageContext이름을 나타낸다.
public static java.lang.String REQUEST	이 마당은 PageContext 이름표에 ServletRequest을 저장하는데 사용된다.
public static int REQUEST_SCOPE	요청범위라고 지정된 이 참조마당은 현재의 요청이 완료될때까지 Servlet이 ServletRequest에서 여전히 사용가능하도록 한다.
public static java.lang.String RESPONSE	이 마당은 PageContext이름표에 ServletResponse를 저장하기 위해 사용된다.
public static java.lang.String SESSION	PageContext이름표에 HttpSession을 저장하기 위해 사용된다.
public static int SESSION_SCOPE	대화접속범위 (이 페지가 대화접속에 참가할때에만 유효하다) – 이 참조마당은 HttpSession에서 Servlet가 무효로 될때까지 사용가능하게 된다.

표 5-3. PageContext의 성원메쏘드

PageContext의 성원메쏘드	
public abstract java.lang.Object findAttribute(java.lang.String name)	주어진 이름의 속성을 page, request, session, application 중에서 찾아 귀환한다.
public abstract void forward(java.lang.String relativeUrlPath)	현재의 ServletRequest와 ServletResponse를 활동중인 다른 응용프로그람에 넘겨준다.
public abstract java.lang.Object getAttribute(java.lang.String name)	page령역안에서 주어진 이름과 련관있는 객체를 귀환한다.
public abstract java.lang.Object getAttribute(java.lang.String name, int scope)	지정된 령역안에서 주어진 이름과 련관있는 객체를 귀환한다.
public abstract java.util.Enumeration getAttributeNamesInScope(int scope)	주어진 령역에서 모든 속성들을 렬거한다.
public abstract int getAttributesScope(java.lang.String name)	주어진 이름으로 정의된 령역을 얻는다.
public abstract java.lang.Exception getException()	례외(exception)객체를 얻는다.
public abstract jspWriter getOut()	out객체를 얻는다.
public abstract java.lang.Object getPage()	page객체를 얻는다.
public abstract javax.servlet.ServletRequest getRequest()	request 객체를 얻는다.
public abstract javax.servlet.ServletResponse getResponse()	response객체를 얻는다.
public abstract javax.servlet.ServletConfig getServletConfig()	config객체를 얻는다.
public abstract javax.servlet.ServletContext getServletContext()	application객체를 얻는다.
public abstract javax.servlet.http.HttpSession getSession()	session객체를 얻는다.

제5절. 내장객체 out

내장객체 out는 앞에서 설명한 pageContext객체의 메쏘드를 사용하여 JSP내부에서 자동생성된다. 의뢰기에 자료를 전송하는데 반드시 필요한 객체이므로 JSP에서 미리 내장객체로 만들어 놓았다. 그 형식은 다음과 같다.

public abstract class jspWriter extends java.io.Writer

jspWriter out

우에서 보는바와 같이 out객체는 jspWriter형이다. 이 jspWriter는 java.io.Writer클라스를 계승받아 생성된 클라스이다. Writer추상클라스를 계승하였으므로 출력기능을 담당하는 write(), print()메쏘드를 사용할수 있다. 방법상 차이는 있지만 JSP의 jspWriter out와 Servlet의 PrintWriter out는 비슷한 역할을 한다.

out객체는 완충기와 관련된 사항만 주의하여 사용하면 된다. 아래의 실례는 완충기의 크기를 설정하고 작업 후 완충기가 얼마나 쓰이였는가를 알아보는 실례이다.

실례 5-6

outTest.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
<%@ page  buffer ="1kb"%>
<H1>out객체검증실례</H1><hr>
<%
  out.println("out객체를 검증하는 실례입니다. "+"<BR>");
  out.println("완충기의 크기:" + out.getBufferSize()+"Bytes<BR>");
  out.println("완충기의 남은 크기:" +out.getRemaining()+"Bytes");
  out.flush();
%>
```

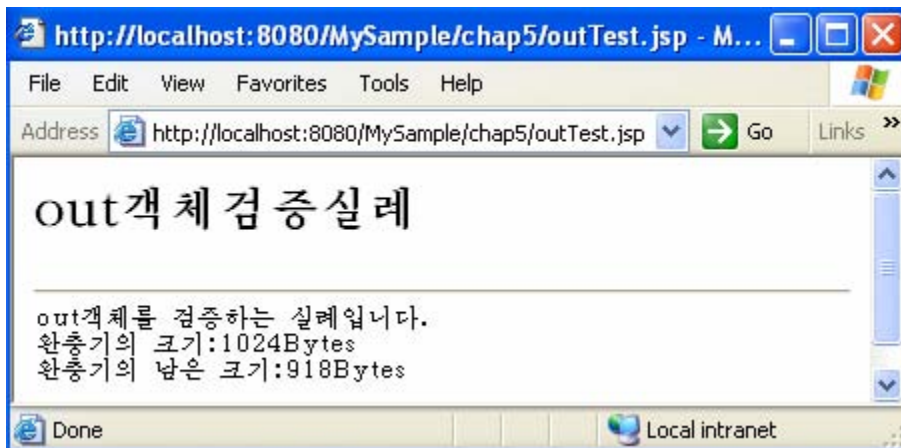



그림 5-11. 실례 5-6의 실행결과

프로그람설명

page지시문을 리용하여 1kbyte로 완충기크기를 설정하고있다. out객체를 리용하여 자료를 흐름에 기록할 때 기록된 자료의 크기가 1kbyte가 되면 자동적으로 의뢰기로 전송되며 다시 완충기에 채워지면 자료가 전송되게 된다.

```
<%@ page  buffer ="1kb"%>
```

여기서 완충기의 자료가 1kbyte가 되지 않은 상태에서 작업을 끝내면 자동적으로 완충기에 있는 자료는 의뢰기로 전송된다. 이것은 다음과 같은 page지시문이 기정으로 설정되였기때문이다.

```
<%@ page  autoFlush ="true"%>
```

getBufferSize()메쏘드는 설정되여있는 완충기의 크기를 얻어내는 메쏘드이다.

```
out.getBufferSize();
```

다음은 완충기의 남은 크기를 귀환하는 메쏘드이다.

```
out.getRemaining();
```

마지막으로 flush()메쏘드를 호출하여 완충기에 들어있는 자료를 출력한다.

```
out.flush();
```

그림 5-11은 이 실례의 실행결과이다.

JSP에서 out는 많이 사용되는 객체이다. JSP에서 어떠한 방식으로 프로그람이 이루어지던 결과는 HTML코드로 변환되여 의뢰기로 전송해야 하므로 의뢰기에 직접 련결된 out가 가장 많이 사용되는것이다. 아래에서 jspWriter의 성원메쏘드들을 소개한다.(표 5-4)

표 5-4. jspWriter의 성원메쏘드들

jspWriter의 성원메쏘드	
public abstract void clear()	완충기의 내용을 지운다.
public abstract void clearBuffer()	현재 완충기의 내용을 지운다.
public abstract void close()	흐름을 닫고 기억기의 내용을 지운다.
public abstract void flush()	흐름을 밀어낸다.
public int getBufferSize()	jspWriter에 의해 사용되는 완충기의 크기를 귀환한다.
public abstract int getRemaining()	완충기에서 사용되지 않은 크기를 귀환한다.
public boolean isAutoFlush()	자동플래쉬기능을 사용하는가를 알려준다.
public abstract void newLine()	새로운 줄에 기록한다.
public abstract void print(boolean b)	boolean값을 출력한다.
public abstract void print(char c)	char값을 출력한다.
public abstract void print(char[] s)	char의 배렬값을 출력한다.
public abstract void print(double d)	double값을 출력한다.
public abstract void print(float f)	float값을 출력한다.

public abstract void print(int i)	int값을 출력한다.
public abstract void print(long l)	long값을 출력한다.
public abstract void print(java.lang.Object obj)	object값을 출력한다.
public abstract void print(java.lang.String str)	string값을 출력한다.
public abstract void println()	빈 행을 생성한다.
public abstract void println(boolean b)	boolean값을 출력하고 행을 바꾼다.
public abstract void println(char c)	char값을 출력하고 행을 바꾼다.
public abstract void println(char[] s)	char의 배렬값을 출력하고 행을 바꾼다.
public abstract void println(double d)	double값을 출력하고 행을 바꾼다.
public abstract void println(float f)	float값을 출력하고 행을 바꾼다.
public abstract void println(int i)	int값을 출력하고 행을 바꾼다.
public abstract void println(long l)	long값을 출력하고 행을 바꾼다.
public abstract void println(java.lang.Object obj)	object값을 출력하고 행을 바꾼다.
public abstract void println(java.lang.String str)	string값을 출력하고 행을 바꾼다.

제6절. 내장객체 application

application내장객체는 javax.servlet.ServletContext대면형의 객체이다. application내장객체의 역할은 다음과 같다.

JSP페지안에서 ServletContext application내장객체의 역할

① 정보관리
- Servlet자체의 정보열람
- 웨브응용프로그람의 전반적인 정보열람
- Servlet엔진 등의 정보열람

② 자료의 저장 및 공유
- ServletContext에 객체 등의 정보를 저장하고 공유
- 다른 Servlet자료의 공유

③ 페지의 조종권한 관리
- 페지의 조종권한을 영구 또는 림시로 넘기는 forward와 include

이렇게 application내장객체는 Servlet 및 웨브응용프로그람자체의 정보를 다룰수 있으며 자료를 저장, 공유하여 여러 곳에서 사용할수 있게 한다. 그리고 pageContext와 같이 페지의 조종권한을 다른 페지로 아주 넘기거나(forward) 잠간 넘겨주는(include) 역할도 수행한다.

앞에서 pageContext내장객체를 정보공유의 측면에서 고찰하였다. 여기서는 ServletContext의 객체인 application내장객체에 대하여 보면서 pageContext내장객체와의 정보공유에서의 공통점과 공유범위의 차이점에 대하여 고찰한다.

정보의 공유측면에서 본 pageContext와 응용프로그람

① pageContext
- 하나의 페지(Servlet)당 pageContext가 하나 존재
- 페지(Servlet)내에서만 정보를 공유

② 응용프로그람
- 하나의 웨브응용프로그람당 Servlet문맥이 하나 존재
- 웨브응용프로그람내에서 정보를 공유

다음의 실례는 Servlet문맥을 리용하여 웨브응용프로그람의 여러 정보 및 Servlet엔진 등의 정보를 알아보는 실례이다. 실행결과는 그림 5-12에서 보여주었다.

실례 5-7

applicationTest.jsp

```
<%@ page contentType= "text/html;charset = BIG5"%>
<%@ page import="java.io.*"%>
<%
  String mimeType = application.getMimeType("java_logo.jpg");
  application.log("******************Log FileTest***********************");
  String serverInfo = application.getServerInfo();
  String realPath = application.getRealPath("applicationTest.jsp");
  int majorVersion = application.getMajorVersion();
  int minorVersion = application.getMinorVersion();
%>
  <B>MimeType  - </B> <%=mimeType%><BR><BR>
  <B>Log File Write ok!!!!!!</B><BR><BR>
  <B>ServerInfo  - </B> <%=serverInfo%><BR><BR>
  <B>RealPath   - </B><%=realPath%><BR><BR>
  <B>MajorVersion -</B>  <%=majorVersion%><BR><BR>
  <B>MinorVersion - </B> <%=minorVersion%><BR><BR>
```

그림 5-12. 실례 5-7의 실행결과

프로그람설명

여기서는 application내장객체를 리용하여 파일의 종류, 등록가입의 기록, 봉사기의 정보, 경로, 엔진의 판본 등을 알아내고있다.

```
String mimeType = application.getMimeType("java_logo.jpg");
application.log("*******************Log FileTest***********************");
String serverInfo = application.getServerInfo();
String realPath = application.getRealPath("applicationTest.jsp");
int majorVersion = application.getMajorVersion();
int minorVersion = application.getMinorVersion();
```

다음은 ServletContext의 메쏘드들을 소개한다.(표 5-5)

표 5-5. ServletContext의 성원메쏘드들

ServletContext의 성원메쏘드	
public java.lang.Object getAttribute(java.lang.String name)	주어진 이름의 속성을 귀환한다.
public java.util.Enumeration getAttributeNames()	속성이름들을 렬거한다.
public ServletContext getContext(java.lang.String uripath)	지정된 봉사기의 URL에서 일치하는 ServletContext객체를 귀환한다.

public java.lang.String getInitParameter(java.lang.String name)	지정된 이름의 초기화변수를 귀환한다.
public java.util.Enumeration getInitParameterNames()	초기화변수의 이름들을 렬거한다.
public int getMajorVersion()	Java Servlet API 의 Major판본을 표시한다.
public java.lang.String getMimeType(java.lang.String file)	지정된 파일의 MIME형을 얻는다.
pulbic int getMinorVersion()	Java Servlet API의 Minor판본을 표시한다.
public RequestDispatcher getNamedDispatcher(java.lang.String name)	주어진 이름의 servlet에서 RequestDispatcher객체를 귀환한다.
public java.lang.String getRealPath(java.lang.String path)	주어진 가상경로를 실제경로로 귀환한다.
public RequestDispatcher getRequestDispatcher(java.lang.String path)	주어진 경로에 위치한 자원(resource)에서 Request Dispatcher객체를 귀환한다.
public java.net.URL getResource(java.lang.String path)	지정된 경로로 넘겨진 자원에 대한 URL을 귀환한다.
public java.io.InputStream getResourceAsStream(java.lang.String path)	주어진 경로의 자원을 InputStream객체형으로 귀환한다.
public java.lang.String getServerInfo() :	동작중인 Servlet용기의 이름과 판본을 귀환한다.
public void log(java.lang.String msg)	지정된 통보문을 Servlet의 일지파일에 기록한다
public void removeAttribute(java.lang.String name)	지정된 이름의 속성을 제거한다.
public void set Attribute(java.lang.String name, java.lang.Object object)	지정된 이름의 속성을 지정된 객체에 추가한다.

제7절. 내장객체 config

config내장객체는 javax.sevlet.ServletConfig대면형의 객체이다.

ServletConfig는 Servlet에 Servlet를 초기화하는 동안 참조해야 할 정보를 전달해주는 역할을 한다. 그러나 JSP는 페지의 요청시에 Servlet가 동적으로 생성되므로 config객체는 Servlet초기화의 메쏘드로는 거의 쓰지 않는다. 다른 방법으로 객체를 얻어 웨브응용프로그람의 정보 등을 알아내는데 사용된다.

여기서는 이러한 config객체의 사용법에 대해 간단히 고찰한다.(그림 5-13)

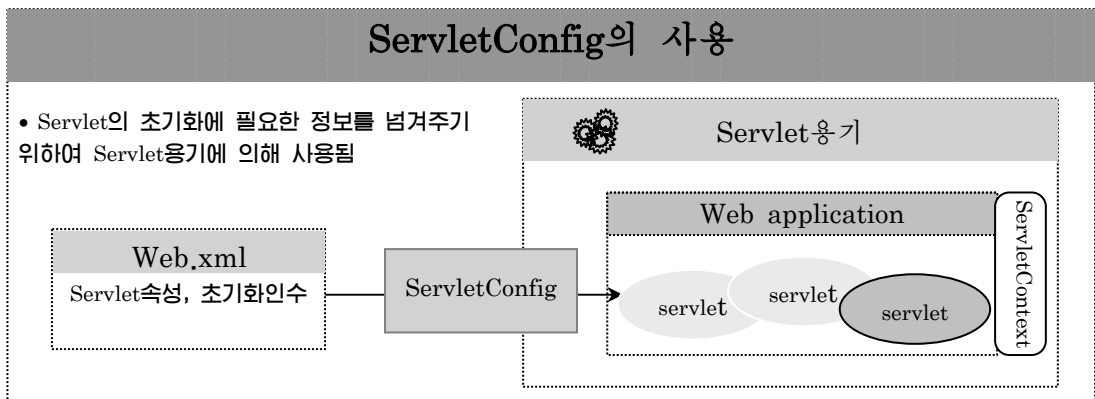



그림 5-13. ServletConfig의 사용

아래의 실례는 config내장객체를 리용하여 봉사기의 정보를 알아보는 실례이다.

실례 5-8

configTest.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<HTML><BODY>
<h2> 내장객체 config를 사용한 정보출력 </h2>
<h3>
<%
  out.println("Servlet 이름: " + config.getServletName()+"<BR><BR>");
  ServletContext context = config.getServletContext();
  out.println("봉사기판본 : "+context.getMajorVersion()+
                        "."+context.getMinorVersion());
%>
</h3><BODY><HTML>
```

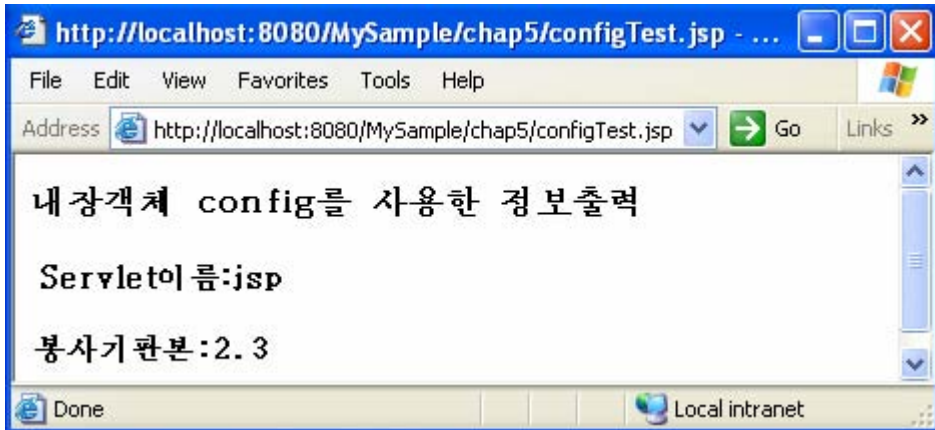



그림 5-14. 실례 5-8의 실행결과

먼저 config객체를 사용하여 Servlet의 이름을 얻고있다. 이 이름은 JSP가 실행되고 Servlet로 변환된 후의 이름이다. 이렇게 생성된 Servlet의 이름은 《jsp이름$jsp》의 형태를 가진다. 그러므로 우의 실례를 실행시키면 단지 《jsp》라고만 Servlet이름을 귀환한다.

out.println("Servlet 이름 : " + config.getServletName()+"

");

다음으로 config를 리용하여 Servlet문맥의 객체를 얻어 봉사기의 정보를 출력하고있다. 그러나 여기서 얻은 ServletContext context는 application내장객체와 동일하므로 이러한 방법으로 ServletContext를 얻을 필요는 없다.

ServletContext context = config.getServletContext();

out.println("봉사기판본:"+context.getMajorVersion()+"."+context.getMinorVersion());

그림 5-14는 이 실례의 실행결과이다.

아래에서 ServletConfig의 성원메쏘드들에 대하여 소개한다.(표 5-6)

표 5-6. ServletConfig의 성원메쏘드들

ServletConfig의 성원메쏘드	
public java.lang.String getInitParameter(java.lang.String name)	주어진 이름의 초기화변수값을 귀환한다.
public java.util.Enumeration getInitParameterNames()	Servlet의 초기화변수이름들을 렬거한다.
public ServletContext getServletContext()	호출자가 실행중인 ServletContext의 참조변수를 귀환한다.
public java.lang.String getServletName()	Servlet객체의 이름을 귀환한다.

제8절. 내장객체 session

session의 원래 클라스형은 HttpSession대면이다. 이 session객체는 사용자가 하나이상의 페지를 요구하거나 다른 거점에로의 방문 또는 저장된 정보에 접근할 때 사용자를 확인해주는 기능을 수행한다.

원래 Http는 비련결상태(stateless)이다. 즉 한번 접속한 후 련결이 유지되지 않고 끊어지므로 접속한 의뢰기나 사용자에 대한 정보를 유지할수 없다. 이와 반대되는 개념으로써 련결상태(stateful)가 있다. 이것은 한번 접속하면 그 접속이 계속 유지된다. 비련결상태를 련결상태로 만들어주는 역할을 수행하는것이 대화접속이다.

대화접속은 봉사기가 가지고있는 일종의 출석부라고 볼수 있다. HTTP봉사기가 HTTP의뢰기나 사용자들을 구분해야 할 경우에는 그것들을 구분할수 있게 매개에 이름표를 달아주어 그 출석부와 이름표를 가지고 상대방의 정보를 유지한다. 대화접속은 지정된 시간동안 또는 사용자가 페지를 계속 요구하는 동안에만 존재한다. 그 외의 경우에 대화접속은 소멸되며 비련결상태로 돌아가게 된다.

- JSP내장객체인 대화접속은 다음과 같이 얻어진다.
  session = pageContext.getSession();
- 대화접속식별자와 식별자에 따르는 값을 얻는 방법은 다음과 같다.
  session.getID();
  session.getValue(“ID”);.

다음의 실례는 대화접속의 유효시간을 설정하고 대화접속의 정보를 얻어내는 실례이다.

실례 5-9

SessionTest.jsp

```
<%@ page contentType="text/html;charset=big5" %>
<HTML>
<BODY>
<% session.setMaxInactiveInterval(60); %>
<h3>Session정보 </h3>
isNew():<%=session.isNew()%><BR>
대화접속ID:<%=session.getId() %><BR>
대화접속생성시간:<%=new java.util.Date
            (session.getCreationTime()).toString() %><BR>
대화접속마지막접속시간: <%=new
java.util.Date(session.getLastAccessedTime()).toString() %><BR>
대화접속Active시간:<%=session.getMaxInactiveInterval() %>sec
</BODY>
</HTML>
```

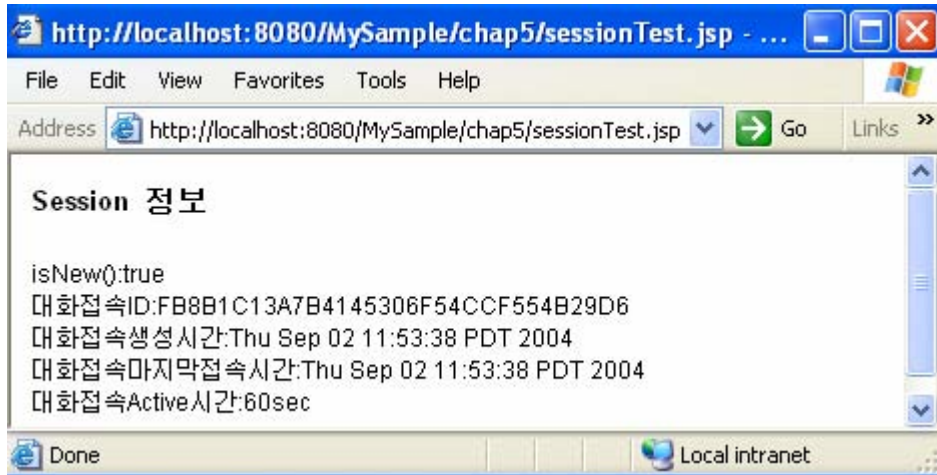



그림 5-15. 실례 5-9의 실행결과

프로그람설명

결과를 보면 대화접속에 관한 다양한 정보가 출력되는것을 알수 있다.

SetMaxInactiveInterval()은 대화접속이 유지될 시간을 설정하는 메쏘드이다. 이 시간이 지나면 대화접속은 자동적으로 완료된다. 단위는 초이다. 여기서는 1분으로 설정하였다. 대화접속유지시간을 5분으로 설정하려면 다음과 같이 한다.

<% session.setMaxInactiveInterval(5*60); %>

isNew()는 대화접속이 처음 생성되였을 경우에는 true를, 이미 생성되여있을 경우에는 false를 귀환한다. 대화접속유지시간이 지난 후 요청을 한다면 다시 true를 귀환한다.

<%=session.isNew()%>

getId()는 이 대화접속에 할당된 유일한 이름을 귀환해주는 메쏘드이다. 이 이름은 Servlet용기가 할당해준다.

<%=session.getId() %>

getCreationTime()은 대화접속이 처음 생성된 시간을 초단위로 귀환해주는 메쏘드이다. 이 실례에서는 Date객체를 사용하여 표준시간으로 변환하였다.

<%=new java.util.Date(session.getCreationTime()).toString() %>

getLastAccessedTime()은 대화접속이 유지되는 동안 마지막에 접속한 시간을 초단위로 귀환해주고있다. 역시 Date객체를 사용하여 표준시간으로 변환하였다.

<%=new java.util.Date(session.getLastAccessedTime()).toString() %>

getMaxInactiveInterval()은 설정된 대화접속 유지시간값을 얻는 메쏘드이다. 즉 setMaxInactiveInterval()의 값을 호출한다. 단위는 역시 초이다.

<%=session.getMaxInactiveInterval() %>

아래에서 HttpSession의 성원메쏘드들에 대하여 소개한다.(표 5-7)

표 5-7. HttpSession의 성원메쏘드들

HttpSession의 성원메쏘드	
public java.lang.Object getAttribute(java.lang.String name)	지정된 이름의 대화접속에 속한 객체를 귀환한다.
public java.util.Enumeration getAttributeNames()	대화접속에 지정된 객체의 이름을 렬거한다.
public long getCreationTime()	1970. 1.1 GMT부터 대화접속이 만들어졌을 때 까지의 시간을 미리초의 단위로 귀환한다.
public java.lang.String getId()	대화접속에 지정된 고유한 이름을 귀환한다.
public long getCreationTime()	대화접속이 처음 생성된 시간을 미리초로 계산하여 long형으로 귀환한다. 기준은 1970년 1월 1일 00시 00분 00초이다.
public long getLastAccessedTime()	의뢰기요청이 마지막으로 진행된 시간을 미리초로 귀환한다.
public int getMaxInactiveInterval()	의뢰기의 요구가 없을 때 봉사기가 현재의 대화접속을 언제까지 유지하겠는가를 정의 옹근수값으로 귀환한다. 이때 기정대화접속마감시간은 30분으로 지정되여있다. 단위는 초이다.
public void invalidate()	현재의 대화접속을 완료시킨다.
public boolean isNew()	봉사기측에서 새로운 session객체를 생성하고 아직 의뢰기에게 대화접속식별자를 할당하지 않은 경우 true를 귀환하고 기정의 대화접속이 유지되고있는 상태라면 false를 귀환한다.
public void setMaxInactiveInterval (int seconds)	대화접속유지시간을 설정한다. 이 시간이 지나면 대화접속은 완료된다.

제9절. 내장객체 page

page객체는 JSP페지에서 생성되는 Servlet객체를 참조하는 참조변수이다. 즉 자기가 생성할 Servlet객체를 참조하는 객체이다. 그러므로 자기자신을 참조하는 열쇠어 this를 사용하여 생성한다.

Object page = this;

page객체는 내부적으로 자기자신을 참조하는 this를 받지만 형이 Object형인것을 볼수 있다. 이렇게 되면 외부에서 page객체를 사용할수 없고 주로 내부적인 작업을 하기 위하여 사용된다. 만일 이 객체를 외부에서 사용하려면 다음과 같이 한다.

실례 5-10

pageTest.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
<H1> page 내장 객체를 검증 </H1><hr>
<%! int sum(int a , int b){
    return a+b;
}
%>
<%
    int aaa = 4, bbb =5;
    out.println("처음 생성된 객체를 리용하여 출력 : " +sum(aaa,bbb)+"<BR>");
    out.println("this 객체를 리용하여 출력 : " +this.sum(aaa,bbb)+"<BR>");
    pageTest$jsp pagetest = (pageTest$jsp)page;
    out.println("page 객체를 리용하여 출력 : "+pagetest.sum(aaa,bbb));
%>
```

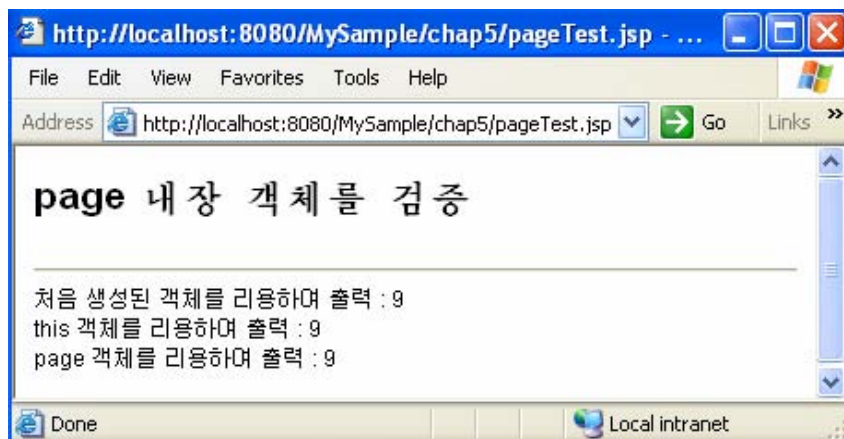



그림 5-16. 실례 5-10의 실행결과

프로그람설명

이 실례에서는 먼저 성원메쏘드를 선언하기 위하여 <%! %>끄리표를 사용하였다. 그리고 성원메쏘드인 sum()메쏘드를 리용하여 aaa와 bbb의 합을 출력한다. sum은 성원메쏘드이므로 this.sum()과 같이 리용할수 있다.

```
out.println("처음 생성된 객체를 리용하여 출력 : " +sum(aaa,bbb)+"<BR>");
out.println("this 객체를 리용하여 출력 : " +this.sum(aaa,bbb)+"<BR>");
```

이제 《Object page = this》라는 명령문으로 자동생성된 page객체를 사용해보자. page객체는 앞에서 설명한것처럼 Object형이므로 구체적인 형을 찾아내야 한다. jsp파일이 Servlet로 변환될 때 Servlet의 이름은 JSP파일의 이름뒤에 $jsp를 붙인 형식이 된다. 결국 이 Jsp파일을 Serlvet로 변환하면 pageTest$jsp.java가 된다.(Tomcat 4.04기준)

pageTest$jsp.java가 콤파일되면 pageTest$jsp.class파일이 된다. 그러므로 pageTest$jsp형으로 강제형전환을 해야 한다. 이때 이 객체이름을 pagetest로 정의한다. 다음 sum()메쏘드를 호출한다. 즉

```
pageTest$jsp pagetest = (pageTest$jsp)page;
out.println("page객체를 리용하여 출력 : "+pagetest.sum(aaa,bbb));
```

이 실례의 실행결과는 그림 5-16에서 보여준다.

제10절. 내장객체 exception

exception객체는 java.lang.Throwable클라스형의 JSP내장객체이다. 이 객체는 이름을 보고도 알수 있듯이 오유처리에 관한 객체이다. 즉 해당 페지의 실행시 Servlet가 처리하지 못한 오유가 발생하면 이 례외를 처리할 페지를 지정하였을 경우 지정된 페지로 례외를 전달하는 역할을 하는 객체이다. 그러나 이런 경우 먼저 례외가 전달될 페지의 isErrorPage라는 속성을 《true》로 정의해야만 사용가능하다. 만일 이 속성이 정의되여있지 않으면 례외객체의 사용은 불가능하다. isErrorPage의 기정값은 false이므로 다음과 같은 지시문으로 isErrorPage속성을 true로 만들어야 한다.

```
<%@ page isErrorPage= "true" >
```

그리고 현재 페지에서 오유가 발생할 경우에는 다음과 같은 형식으로 오유를 처리할 JSP페지를 지정한다. 이때 주의할 점은 buffer속성을 false로 설정하여서는 안된다는것이다. 례외정보를 가지고 넘어가므로 저장할 완충기공간이 필요하기때문이다.

```
<%@ page errorPage= "상대적URL을 지정(실례로 calculerr.jsp)" %>
```

우에서 설명한 내용을 아래의 실례와 련관시켜 표현하면 그림 5-17과 같다.

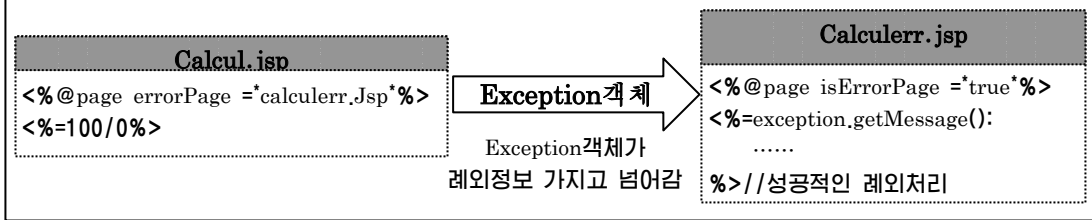



그림 5-17. exception의 례외처리

다음의 실례는 홈양식으로부터 값을 받아 사칙연산을 진행하는 JSP프로그람이다. 계산시 오유가 발생하였을 경우 오유페지로 넘겨 처리하도록 례외처리페지를 만들어준다.

실례 5-11

calcul.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<!--<%@ page errorPage = "calculerr.jsp" %> -->
<%
String first = request.getParameter("first");
String last = request.getParameter("last");
String oper = request.getParameter("oper");
if (first == null || last == null){
%>
<HTML><BODY>
<h2>Calculator</h2>
<h3>수자를 입력하십시오</h3>
<form action = "calcul.jsp" method = "Post">
<input name = "first" size = "10">
<input type=radio name="oper" value="plus" checked>+
<input type=radio name="oper" value="minus">-
<input type=radio name="oper" value="multi">*
<input type=radio name="oper" value="divide">/
<input name = "last" size = "10">
<input type="submit" value="계산">
</form>
</BODY></HTML>
 <%
} else {
  int first1 = Integer.parseInt(first);
  int last1 = Integer.parseInt(last);
  String operator = null;
  int result = 0;
  if (oper.equals("plus")) { result = first1 + last1; operator ="+";}
  if (oper.equals("minus")) { result = first1 - last1; operator ="-";}
```

```
  if (oper.equals("multi")) { result = first1* last1; operator ="*";}
  if (oper.equals("divide")) { result = first1 / last1; operator ="/";}
  out.println("<h2>계산결과<h2>");
  out.println("<h2>" + first1 + operator + last1 + "=" + result + "<h2>");
}
%>
```

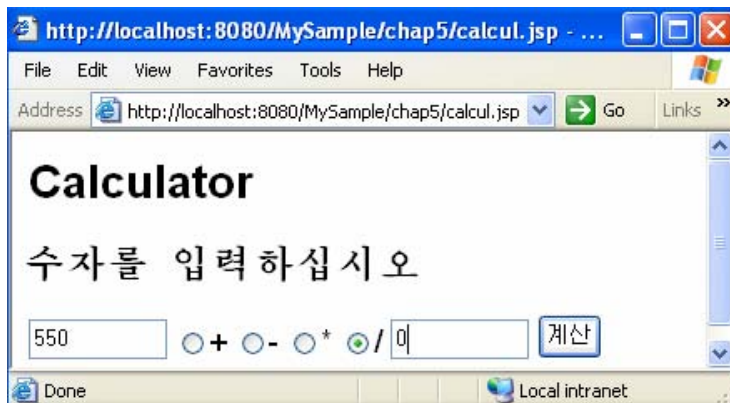
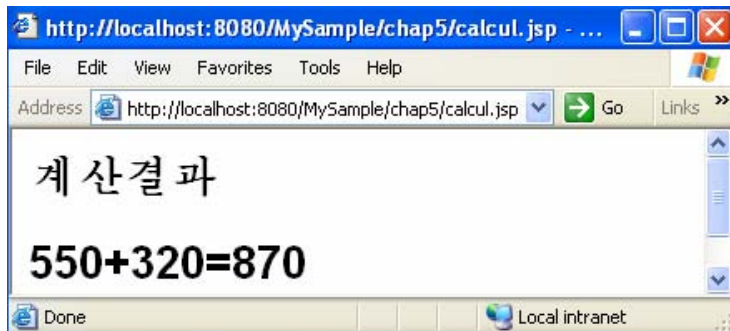
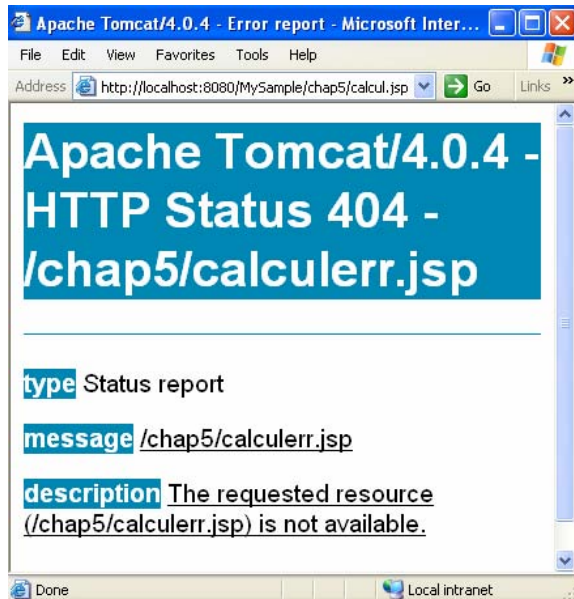



그림 5-18. 실레 5-11의 실행결과

프로그람설명

우의 실례를 보면 HTML홈양식으로 간단하게 사칙연산의 대면을 만들고 JSP스크립트부분에서 홈의 Post방식으로 넘어온 자료를 받아서 계산하는것을 볼수 있다. 먼저 결과를 보면(그림 5-18) 《550+320=870》으로 덕셈의 결과는 잘 나왔지만 《550/0》의 연산을 실행시키면 그림 5-19와 같이 아주 복잡한 오유발생코드가 의뢰기에 보이게 된다. 우와 같은 현상을 방지하기 위하여 아래와 같은 코드를 만들어준다.

```
<%@ page errorPage = "calculerr.jsp" %>
```

그림 5-19. 오유발생통보

그리고 아래와 같이 오유를 처리해주는 calculerr.jsp페지를 만든다. 실행결과는 그림 5-20에서 보여준다.

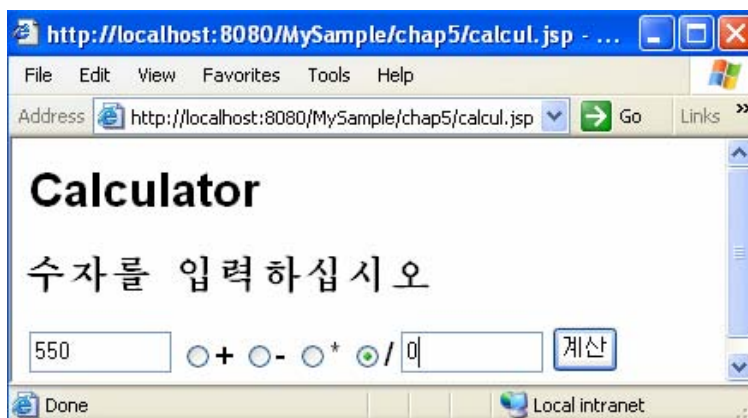



그림 5-20. 실례 5-12의 실행결과

실례 5-12

calculerr.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<%@ page isErrorPage = "true" %>
<HTML>
<BODY>
<h2>오유처리 페지</h2>
<h3>다음 문제 때문에 오유가 발생하였습니다.</h3>
<I><%= exception.getMessage()%></I>
<h3>오유의 형식은 다음과 같다.</h3>
<I><%= exception %></I>
</BODY>
</HTML>
```

프로그람설명

이 원천과 같이 오유를 처리해줄 페지를 만들면 오유가 발생하는 경우 calcul.jsp페지에서 《<%@ page errorPage= "상대적URL" %>》를 통하여 exception객체가 오유정보를 가지고 calculerr.jsp의 페지에로 넘어가게 된다.

그러면 이 페지에서는 현재 페지가 오유를 처리할수 있는 상태라는것을 다음과 같은 형식으로 표시한다.

```
<%@ page isErrorPage = "true" %>
```

그리고 exception객체를 사용하여 사용자에게 보여줄 오유페지를 적당히 설정한다.

```
<h3>다음 문제 때문에 오유가 발생하였습니다.</h3>
<I><%= exception.getMessage()%></I>
```

<h3>오유의 형식은 다음과 같다.</h3>

<I><%= exception %></I>

만일 isErrorPage속성을 false로 하였을 경우에는 오유페지는 자기의 기능을 수행하지 못한다.

아래에서 Throwable의 성원메쏘드들에 대하여 소개한다(표 5-8).

표 5-8. java.lang.Throwable의 성원메쏘드들

java.lang.Throwable의 성원메쏘드
public String getMessage() : 례외객체의 오유통보문을 문자렬형으로 돌려준다. public void printStackTace() : 표준오유출력흐름으로 오유경로를 추적하여 출력해준다. public void printStackTrace(PrintStream s) : 정해진 출력흐름으로 오유경로를 추적하여 출력해준다.

제11절. 지시문

5.11.1. 지시문의 종류

지시문은 JSP페지안에서 속성을 지정하거나 현재의 페지에서 작업하게 되는 내용을 가리키는 끄리표이다. 지시문은 크게 3가지로 분류할수 있다.

지시문에서 사용되는 형태

```
<%@page %>
<%@include %>
<%@taglib %>
```

• page지시문은 JSP페지의 속성을 지정하는데 사용한다. JSP에서 지정할수 있는 속성은 language, import, extends 등 총 11개가 있다. 이것들에 대하여서는 다음 절에서 고찰한다. page지시문의 형식은 다음과 같다.

```
<%@page attribute1= “value1” attribute2= “value2” attribute3= “value3” ... %>
```

• include지시문은 말그대로 지정한 파일을 JSP파일에 삽입하는데 사용한다. 지시문의 include는 pageContext와 응용프로그람의 include와 약간 다른 측면을 가지고있다. pageContext와 응용프로그람의 include는 조종권한과 관련되지만 지시문의 include는 문서의 단순한 포함을 의미하며 조종권한과는 관련이 없다. 즉 단순히 파일만을 포함할 때 include지시문을 리용한다. 파일이 정적으로 포함되므로 jsp파일안의 명령문이 그대로 현재의 페지에서 실행된다. 그의 사용방법은 다음과 같다.

```
<%@include file = “include.jsp(html)” %>
```

만일 지시문에 jsp파일을 포함하였을 때 JSP내용들이 실행되는것이 아니라 원천이 그

대로 이 페지에 출력된다. 이 점에 주의하여야 한다.

- taglib지시문은 JSP페지에서 사용자정의꼬리표(custom tag)를 사용한다는것을 Servlet용기에 알려주는 역할을 한다. 즉 taglib지시문에는 XML을 기반으로 하는 사용자정의꼬리표를 정의하는 꼬리표서고서술파일(TLD: Tag Library Descriptor)이 위치한 URI를 지정해준다. taglib의 형식은 TLD이 존재하는 URI를 지정해 준 다음 prefix로 꼬리표의 식별자를 지정하여 사용한다. 실례로

<%@tablib uri = “MySample/sampletag” prefix = “samplePrefix” %>

5.11.2. page지시문

page지시문에 대하여 고찰한다. page지시문은 페지에 대한 속성을 지정하는데 사용한다. 일반적인 형식은 다음과 같다.

<%@ page attribute1="value" attribute2="value"%>

《속성=값》의 형식으로 page지시문에서 속성을 지정한다. page지시문안에서 사용하는 속성들은 다음과 같다.(표 5-9)

표 5-9. Page지시문에서 사용가능한 속성들

속성명	설 명	기정값
language	스크립트안에서의 언어명	language="java"
contentType	페지의 MIME형지정	"text/html;charset=ISO-8859-1"
info	해당페지에 대한 설명문	없음
extends	상위클라스지정	없음
import	class, package적재	없음
session	대화접속의 사용여부 결정	session = "true"
buffer	완충기의 크기결정	buffer="8kb"
autoFlush	자동플래쉬기능설정	autoFlush = "true"
isThreadSafe	SingleThreadModel대면의 실현여부 결정	isThreadSafe ="true"
errorPage	오유페지를 지정	없음
isErrorPage	오유페지의 사용여부 결정	isErrorPage = "false"

매 속성들에 대하여 하나씩 보기로 하자.

- **language**

JSP페지의 프로그람작성언어를 지정한다. 기정값은 java이며 현재 쓰일수 있는 유일한 값도 java이다.

<%@page language = “java” >

- contentType

의뢰기에게 응답할 문서의 MIME형을 지정한다. 그 형식을 보면 type에는 응답할 형을 지정하고 추가로 부호화방식이 필요할 때에는 charset란 속성을 써주면 된다.

<% page contentType = "text/html; charset = big5">

text/html이외에 Servlet의 기본류형인 text/plain, 화상을 위한 image/gif 등의 류형도 있다.

- extends

JSP페지를 요청할 때 생성되는 Servlet의 상위클라스를 지정하는데 쓰인다. 그러나 Servlet용기는 용기개발회사에서 이미 만들어진 최적의 상위클라스를 사용하여 동적으로 Servlet를 생성하는 경우가 대부분이다. 따라서 사용빈도가 거의 없는 속성이라고 볼수 있다.

<%@page extends = "father.class" %>

- info

해당 페지에 대한 설명을 지정할 때 쓰이는 속성으로서 이것을 지정하면 getServerInfo()메쏘드로 지정한 정보를 알아볼수 있다.

<%@ page info = "소설JSP info페지" %>

- import

JSP가 요청되여 변환된 Servlet가 적재할 클라스를 지정한다. 즉 Servlet가 기본적으로 적재하는 java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.* 외에 필요한 패키지를 적재하려는 경우 import를 리용한다. 사용형식은 다음과 같다.

<%@page import = "java.util.*" %>
<%@page import = "chap4.*, chap5.*" %>

- session

페지가 HttpSession을 사용하겠는가 하는것을 지정한다. 속성은 true와 false로 되며 true일 경우에는 대화접속이 이미 존재할 경우 그 대화접속을 유지하고 존재하지 않을 경우는 새로운 대화접속을 생성하여 련결된다. false일 경우에는 대화접속이 련결되지 않는다. 기정값은 true이다.

<%@page session = "true" %>
<%@page session = "false" %>

- buffer

의뢰기에로의 전송을 담당하는 out객체(jspWirter out)의 완충기크기를 설정한다. 완충기크기를 설정하면 지정된 완충기 크기만큼의 자료를 단위로 의뢰기에 전송된다. 즉 완충기의 크기가 30kbyte라면 완충기는 30kbyte의 자료를 채우기 전까지는 의뢰기에 전송하지 않고 30kbyte가 되는 순간 또는 30kbyte가 안되더라도 페지처리가 완료되는 순간 의뢰기로 자료를 전송하게 된다.

<%@page buffer = "30kb" %>
<%@page buffer = "none" %>

우에서 none으로 한 경우는 완충기를 사용하지 않는다는것을 의미한다.

- autoflush

출력완충기가 꽉 찼을 때 이를 조종하는 방법을 지정한다. true로 지정된 경우에는 출력완충기의 내용을 의뢰기에 보내고 완충기는 비게 된다. false로 지정한 경우에는 완충기의 내용을 비우지 않으므로 례외를 발생시킨다. 만일 완충기의 크기를 none으로 지정했다면 autoflush속성을 true로 지정할수 없다. 이 속성의 기정값은 true이다.

<%@page autoflush = "true" %>
<%@page autoflush = "false" %>

- isThreadSafe

JSP에서 생성된 Servlet가 SingleThreadModel대면을 실현하는가 하는것을 지정한다. SingleThreadModel은 4장에서 이미 고찰하였다. Servlet에서 스레드로 자원에 접근할 때 공유로 인한 문제점이 있을 경우 한번에 하나의 스레드만 자원에 접근하도록 해주는 표시(marker)대면이다. 기정값은 true로 되여있는데 SingleThreadModel을 실현하려면 값을 false로 지정해야 한다.

<%@page isThreadSafe = "true" %>
<%@page isThreadSafe = "false" %>

- errorPage

페지에서 어떠한 례외상황이 발생할 경우 그 오유를 처리할 페지를 지정한다. 이렇게 지정된 페지로 례외객체가 오유를 가지고 넘어간다.

<%@page errorPage= "errManage.jsp" %>

- isErrorPage

페지가 다른 페지의 오유처리페지로 되는가 하는 여부를 지정한다. 즉 우에서 errorPage속성으로 례외를 넘겼을 때 그것을 받아서 처리할것인가 하는것을 결정한다. 기정값은 false이다.

<%@page isErrorPage= "true" %>
<%@page isErrorPage= "false" %>

5.11.3. include지시문

include지시문은 지정된 파일을 JSP페지안에 삽입할 때 쓰이는 지시문이다. 이것의 특징은 정적(static)이라는것이다. 이 의미를 동적(Dynamic)이라는 의미와 비교하여 고찰해보자.

앞에서 다른 페지를 포함하는 메쏘드로서 pageContext의 include에 대하여 보았다. pageContext의 include는 요청과 조종권한을 잠시 다른 페지로 넘겨주고 그곳에서 동적으로 처리한 결과를 다시 받아서 처리한다. 아래의 표(표 5-10)에서 자세히 보도록 하자.

표 5-10. Include지시문

구분	pageContext의 include	include지시문
형식	pageContext.include ("included.jsp");	<%@ include file = "include d.jsp" %>
처리방법	동적 * 포함될 페지에서 처리하여 결과만을 읽어들여 포함시킨다. * 포함될 페지가 이미 콤파일되여 있어야 한다.	정적 * 포함될 페지를 그대로 본문으로 읽어들여 원래 페지에 삽입한 후 처리한다. * 포함될 페지를 콤파일하였는가 하지 않았는가에 관계없다.
요청 및 조종권한	포함시킬 페지로 잠시 넘긴다.	전혀 상관없다.(정적)

include지시문을 리용하여 다른 파일을 읽어들일 때에는 다른 파일에 기록되여있는 본문내용을 읽어들여 호출한 페지에 그대로 삽입하여 넣는다.

그러면 실례를 통하여 같은 페지를 각각 pageContext의 include와 include지시문을 리용하여 삽입하고 그 차이점을 살펴보도록 하자.

실례 5-13

staticInclude.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<h2>include지시자</h2>
<%String s = "included OK!";%>
<%@include file = "included.jsp" %>
```

dynamicInclude.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<h2>pageContext.include</h2>
<%String s = "included OK!";
pageContext.include("included.jsp");
%>
```

included.jsp

```
<H1><%= s %></H1>
```

그림 5-21. 실례 5-13의 실행결과

프로그람설명

실례를 보면 included.jsp파일을 staticInclude.jsp와 dynamicInclude.jsp에서 각각 포함하고있다. included.jsp는 단지 s라는 변수를 출력해주는 한 행의 코드로 되여있는데 그것만 별도로 실행시키면 결과에서와 같이 오유가 발생한다. 그것은 변수를 선언도 하지 않고 사용하였기때문이다.

staticInclude.jsp에서는 지시문을 통하여 included.jsp를 포함한다.
<%@include file = "included.jsp" %>
결과를 보면 String으로 입력한 변수값이 출력되였음을 알수 있다.(그림 5-21의 두번째 그림)

dynamicInclude.jsp에서는 pageContext.include()를 통하여 included.jsp를 포함한다.(그림 5-21의 세번째) 물론 included.jsp를 그대로 실행하면 String변수 s에 값이 없으므로 귀환값을 돌려줄수 없고 그림 5-21의 첫번째와 같은 결과가 나타난다.

이렇게 include지시문을 리용하여 파일을 삽입하면 그 파일이 본문 그대로 삽입된다는것을 알수 있다.(그림 5-21)

아래에서 include메쏘드를 활용한 실례를 하나 더 고찰한다. 이 실례는 하나의 jsp페지에서 jsp파일, txt파일, html파일을 포함시켜 출력하는 실례이다. 실행결과는 그림 5-22에서 보여준다.

실례 5-14

multiInclude.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<%
  pageContext.include("header.jsp");
  pageContext.include("body.txt");
  pageContext.include("footer.html");
%>
<BR><BR>
```

header.jsp

```
<%@ page contentType="text/html; charset=big5" %>
<h2>Multi included header</h2>
<%String s = "included OK!";
pageContext.include("included.jsp");
%>
```

body.txt

```

txt파일부분이다. 임의의 내용을 포함시키십시오.

```

footer.html

```
<BR><BR>
<hr>
<h3> multi include footer</h3>
HTML파일 부분이다.<BR>
Copyright @2002 All right reserved
</BODY>
</HTML>
```

그림 5-22. 실례 5-14의 실행결과

프로그람설명

multiInclude.jsp를 보면 include메쏘드를 통하여 아래에 서술해놓은 3개의 파일 header.jsp, body.txt, footer.html을 포함시키고있다.

```
pageContext.include("header.jsp");
pageContext.include("body.txt");
pageContext.include("footer.html");
```

삽입될 jsp, txt, html파일들은 각각 통보문을 가지고있다. 결과를 보면 매개 페지의 내용들이 삽입된 순서대로 출력되였다는것을 알수 있다. 이렇게 include메쏘드를 리용하면 파일종류에 관계없이 본문으로 이루어진것이라면 내용 그대로 현재의 파일에 삽입할수 있다. 또한 하나를 만들어놓고 여러 곳에서 재사용할수 있다.

제12절. 액션끄리표

JSP에서 Java원천코드를 삽입하기 위하여 스크립트끄리표(<% %>, <%! %>, <%= %>, <%@ %>)를 리용하였다. 그러나 이 방법외에도 액션끄리표를 리용하는 방법이 있다. 스크립트끄리표를 리용할 때 몇몇 기능을 XML기반의 새로운 끄리표를 만들어서 사용하게 되는데 이것을 **액션끄리표**라고 한다. JSP에서 사용하는 액션끄리표들중에서 대표적인것은 다음과 같다.

jsp:useBean
jsp:setProperty
jsp:getProperty
jsp:include
jsp:forward
jsp:plugin
jsp:param

여기에서 jsp:useBean, jsp:setProperty, jsp:getProperty는 1장에서 간단히 설명하였다.

5.12.1. JSP Bean을 만드는 방법

JSP에서 Bean을 만드는 방법은 아주 간단한다.

Bean을 만들 때 일반적인 Java파일과 같은 방식으로 만들면 된다. jsp:bean끄리표를 사용하는 JSP Bean은 다음의 간단한 규칙을 지켜서 만들어야 JSP Bean으로 사용할수 있다. 그 규칙은 다음과 같다.

① 클라스안의 성원마당은 소문자로 시작한다.

② 성원마당에 값을 설정하거나 얻는 메쏘드는 공개(public)메쏘드로 되여야 한다.

③ 값을 설정하는 메쏘드의 이름은 《set+대문자》로 시작하는 성원마당이름이여야 한다.

④ 값을 얻는 메쏘드의 이름은 《get+대문자》로 시작하는 성원마당이름이여야 한다.

이러한 규칙을 지켜 Bean을 만든다면 JSP의 Bean으로 쓸수 있다. 물론 일반 Java프로그람처럼 프로그람을 작성해도 그 기능들은 모두 사용할수 있다. 우의 규칙을 지키면서 JSP Bean을 만들어 보자.

실례 5-15

ActionTagTest.java

```
package chap5;
public class ActionTagTest{
   private String str="";
   public String getStr(){
      return str;
   }
   public void setStr(String str){
      this.str=str;
   }
   private String id="";

   public void setId(String id){
      this.id=id;
   }
   public String getContent(){
      return id + ":" + str;
   }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap5>javac ActionTagTest.java

프로그람설명

JSP의 Bean으로서 사용하기 위하여 성원마당(String str)은 소문자로 만들었다. 그리고 str을 설정하거나 얻는 메쏘드는 규칙에 맞게 각각 setStr, getStr으로 만들었다. 이러한 규칙을 성원마당 id에도 적용하여 setId로 설정하였다. 그리고 일반적인 Java파일의 성원메쏘드 getContent를 만들었다.

- JSP Bean의 규칙을 지키면서 만든 성원마당과 성원메쏘드

str → getStr(), setStr(String str)

id → setId(int id)

- 일반적인 성원메쏘드

getContent()

이렇게 만들어진 Bean을 JSP에서 사용하는 방법에 대하여 고찰한다.

5.12.2. JSPBean을 생성하는 방법

앞에서 만든 ActionTagTest.class라는 Bean을 리용하여 JSP에서 Bean을 생성해보자. 이때 2가지 방법으로 Bean을 생성할수 있다. 즉

- new를 사용하여 생성하는 방법
- jsp:useBean액션끄리표를 리용하는 방법

먼저 new를 리용하여 Bean을 생성하는 방법에 대하여 고찰한다.

실례 5-16

NewTest.jsp

```
<%@page import="chap5.*" %>
<HTML><BODY><font size=5 color=blue>
<%
  ActionTagTest t1=new ActionTagTest();
  t1.setStr("JSP");
  t1.setId("javabook");
  out.print(t1.getStr());
%><BR>
<%= t1.getContent()%>
</font></BODY></HTML>
```

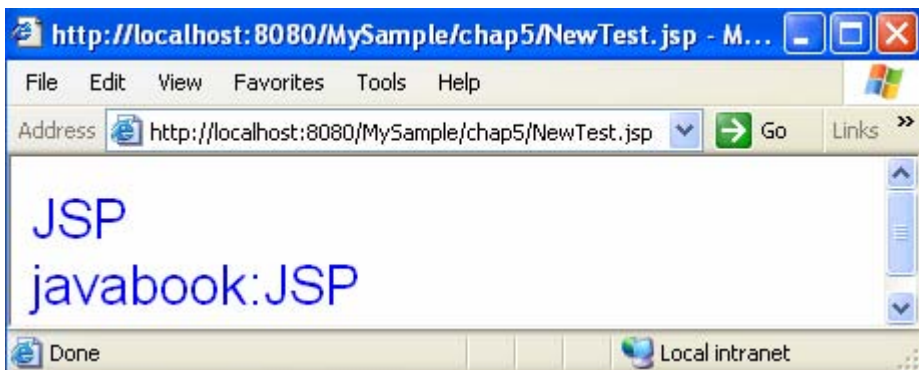



그림 5-23. 실례 5-16의 실행결과

프로그람설명

이 실례에서는 ActionTagTest클라스를 사용하여 Bean을 생성하였다.

먼저 chap5에 있는 모든 클라스파일을 적재하고있다. 그리고 new연산자를 리용하여 ActionTagTest클라스로부터 객체 t1을 생성한다.

```
<%@page import="chap5.*" %>
```

ActionTagTest t1=new ActionTagTest();
set메쏘드를 사용하여 변수 str와 id에 값을 설정한다.
t1.setStr("JSP");
t1.setId("javabook");
get메쏘드를 사용하여 변수 str와 id의 값을 얻는다.
<% out.print(t1.getStr()); %>
<%= t1.getContent()%>
실행결과는 그림 5-23에서 보여주었다.
다음은 액션끄리표를 리용하여 Bean을 생성해 본다. 실행결과는 그림 5-24와 같다.

실례 5-17

ActionTagTest.jsp

```
<HTML><BODY><font size=5 color=blue>
<jsp:useBean id="aTag" class="chap5.ActionTagTest" />
<jsp:setProperty name="aTag" property="str" value="Hello!"/>
<jsp:setProperty name="aTag" property="id" value="javabook"/>
<jsp:getProperty name="aTag" property="str"/><BR>
<jsp:getProperty name="aTag" property="content" />
</font></BODY></HTML>
```

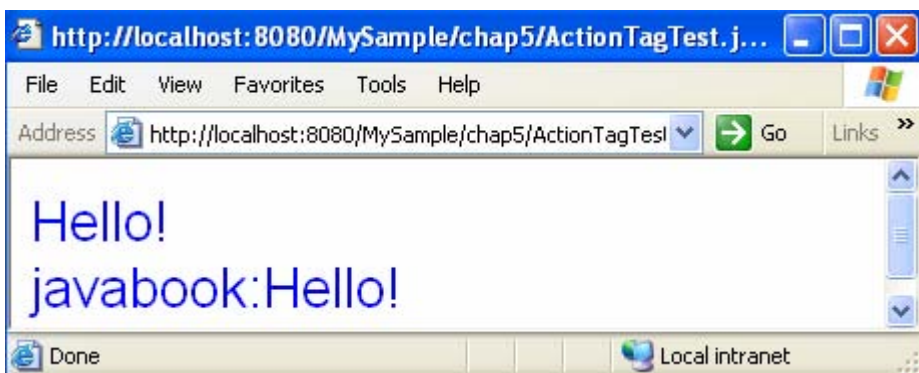



그림 5-24. 실례 5-17의 실행결과

프로그람설명

이 실례에서는 Bean을 생성하기 위하여 다음과 같은 액션끄리표를 사용하였다.
<jsp:useBean id="actTag" class="chap5.ActionTag.ActionTagTest"/>
JSP Bean의 규칙에 맞추어 Bean을 만들었으면 setProperty와 getProperty를 다음과 같이 사용할수 있다.

```
<jsp:setProperty name="aTag" property="str" value="Hello!"/>
<jsp:setProperty name="aTag" property="id" value="javabook"/>
<jsp:getProperty name="aTag" property="str"/><BR>
<jsp:getProperty name="aTag" property="content" />
```

이 명령문들은 JSP Bean객체의 setStr, getStr, setId와 동일한 역할을 한다. setId는 한쪽에만 존재하므로 getProperty액션끄리표를 사용할수 없다. set메쏘드와 get메쏘드는 항상 쌍으로 만들어 주지 않아도 된다. 필요한 속성이 읽기전용이라면 get메쏘드만을, 쓰기전용이라면 set메쏘드만, 읽기와 쓰기가 모두 필요하다면 get과 set를 모두 만들면 된다.

5.12.3. jsp:useBean의 활용

앞에서는 스크립트레트에서 객체를 처리하거나 액션끄리표에서 객체를 처리하는 방법에 대하여 고찰하였다. 여기서는 이 두 방법을 혼합한 실례에 대하여 보기로 한다.

실례 5-18

ActionTagTest.jsp

```
<HTML><BODY><font size=5 color=blue>
<jsp:useBean id="actTag" class="chap5.ActionTagTest" />
<jsp:setProperty name="actTag" property="str" value="MixTest" />
<jsp:getProperty name="actTag" property="str" /><BR>
<%
    actTag.setStr("안녕하십니까!");
    actTag.setId("김은철동무:");
    out.print(actTag.getStr());
%>
<jsp:getProperty name="actTag" property="content"/>
</font></BODY></HTML>
```

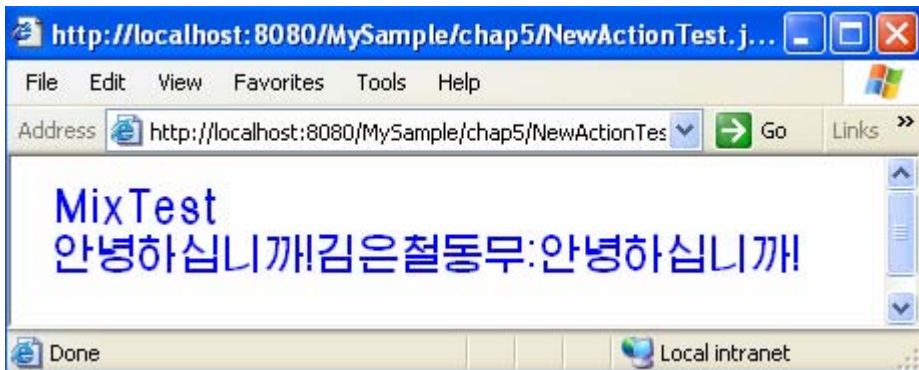



그림 5-25. 실례 5-18의 실행결과

프로그람설명

우의 실레에서는 액션끄리표와 스크립트레트를 함께 사용하였다.

우선 적재한 클라스로부터 객체를 생성한다.

```
<jsp:useBean id="actTag" class="chap5.ActionTagTest" />
```

변수 str에 setProperty와 getProperty를 사용하였다.

```
<jsp:setProperty name="actTag" property="str" value="MixTest" />
<jsp:getProperty name="actTag" property="str" /><BR>
```

액션끄리표로 생성한 actTag를 스크립트레트에서 그대로 사용하였다. setStr()와 setId()메쏘드를 사용하여 변수에 값을 설정하고 getStr()메쏘드를 사용하여 출력하고있다. 마지막에는 다시 액션끄리표를 사용하여 값을 출력하고있다.

```
<%
actTag.setStr("Hi~!");
actTag.setId("Tom:");
out.print(actTag.getStr());
%><BR>
<jsp:getProperty name="actTag" property="content"/>
```

이로부터 액션끄리표와 스크립트레트를 함께 사용하여도 규칙만을 지킨다면 사용하는데 아무런 문제가 없다는것을 알수 있다.

5.12.4. 액션끄리표를 리용한 초기화

이번에는 액션끄리표를 사용하여 Bean을 초기화하는 방법에 대하여 고찰한다.

스크립트레트에서는 구성자를 사용하여 Bean을 초기화할수 있다. 이때 구성자에서 사용되는 파라메터의 유무에는 무관계하다. 따라서 입력된 파라메터를 통하여 Bean의 초기화가 가능하다.

그러나 액션끄리표에서는 반드시 파라메터가 없는 구성자만을 사용하여야 Bean을 생성할수 있다. 즉 구성자의 파라메터를 통하여 Bean을 초기화하는것은 불가능하다. 이것을 극복하기 위하여 액션끄리표에서는 다른 방법을 사용한다. 그 방법은 useBean끄리표안에 Bean의 초기화를 위한 코드를 삽입하는것이다. 여기에 삽입된 코드는 구성자처럼 Bean이 초기에 생성될 때 단 한번만 수행된다.

그러면 실제로 실례를 통하여 스크립트레트를 사용한 경우와 액션끄리표를 사용한 경우를 비교해보자.

실례 5-19

ConscriptletTest.java

```
package chap5;
public class ConscriptletTest{
        private String id="";
        private String str="";
        public String getContent(){
                return id+":"+str;
        }
        public void setStr(String str){
                this.str=str;
        }
        public void setId(String id){
                this.id=id;
        }
        public ConscriptletTest(){
                this.id="javabook";
                this.str="not";
        }
        public ConscriptletTest(String id, String str){
                this.id=id;
                this.str=str;
        }
}
```

C:\Tomcat\webapps\MySample\WEB-INF\classes\chap5>javac ConscriptletTest.java

프로그람설명

먼저 구성자를 검증하기 위한 Bean을 작성하였다. 이 Bean에는 2개의 구성자가 있다. 즉 파라메터가 없는것과 파라메터가 2개있는 구성자가 있다. 파라메터가 없는 구성자에서는 id와 str값을 javabook와 not로 초기화하고있다.

```
public ConscriptletTest(){
this.id="javabook";
this.str="not";
}
```

파라메터가 2개있는 구성자에서는 id와 str을 파라메터로 받아서 초기화하고있다.

```
public ConscriptletTest(String id, String str){
this.id=id;
this.str=str;
}
```

실제로 스크립트레트에서 우의 두 경우를 모두 사용해보자.

실례 5-20

ConscriptletTest.jsp

```
<%@page import="chap5.*" %>
<HTML><BODY><font size=5 color=blue>
<%
   String id=request.getParameter("id");
   String str=request.getParameter("str");
   ConscriptletTest t1=new ConscriptletTest(id,str);
   out.print(t1.getContent());
%><BR>
<%
   ConscriptletTest t2=new ConscriptletTest();
   out.print(t2.getContent());
%>
</font></BODY></HTML>
```

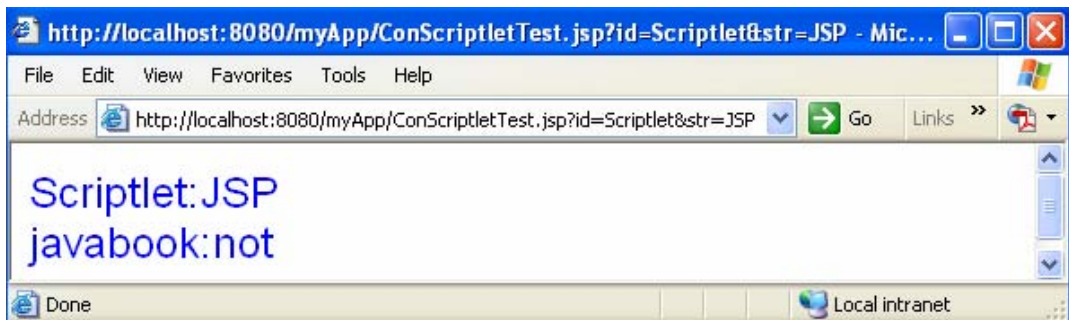



그림 5-26. 실례 5-20의 실행결과

프로그람설명

의뢰기로부터 전송된 id와 str값을 스크립트레트에서 생성한 지역변수 id와 str에 넣어준다.

String id=request.getParameter("id");

String str=request.getParameter("str");

우에서 주어진 값을 구성자의 파라메터로 하여 Bean을 생성한다. ConscriptletTest t1=new ConscriptletTest(id,str);

getContent()메쏘드를 사용하여 초기화된 변수 id와 str의 값을 확인하기 위하여 출력한다. 그 결과 《Scriptlet:JSP》가 출력되였다.

두번째로 파라메터가 없는 구성자를 사용하여 Bean을 초기화하였다. id에는 javabook, str에는 not가 각각 할당되였다.

ConscriptletTest t2=new ConscriptletTest();

getContent()메쏘드를 사용하여 값을 출력하여 확인하였다. 그림 5-26에서 두번째 줄과 같이 출력되는것을 확인할수 있다.

out.print(t2.getContent());

다음은 액션끄리표를 사용하여 Bean을 초기화하는 실례를 보기로 하자.

실례 5-21

ConActionTest.jsp

```
<HTML><BODY><font size=5 color=blue>
<jsp:useBean id="cs1" class="chap5.ConscriptletTest"/>
<jsp:getProperty name="cs1" property="content" />
<BR>
<jsp:useBean id="cs2" class="chap5.ConscriptletTest"/>
<jsp:setProperty name="cs2" property="*" />
</jsp:useBean>
<jsp:getProperty name="cs2" property="content" />
</font></BODY></HTML>
```

그림 5-27. 실례 5-21의 실행결과

프로그람설명

jsp:useBean액션끄리표는 반드시 파라메터가 없는 기정구성자로만 Bean을 생성한다.

먼저 jsp:useBean액션끄리표를 사용하여 Bean을 생성한다. 여기에는 jsp:useBean끄리표내부에 초기화코드를 사용하지 않았으므로 ConscriptletTest클라스의 파라메터가 없는 구성자가 실행된다. 따라서 id와 str은 각각 javabook와 not로 초기화된다.

<jsp:useBean id= "cs1" class= "chap5.ConscriptletTest" />

초기화된 값을 확인하기 위하여 getContent()메쏘드를 사용한다. 이때

《jabook:not》가 출력되는것을 확인할수 있다.(그림 5-27)

<jsp:getProperty name= "cs1" property= "content" />

이번에는 jsp:useBean액션끄리표내부에서 Bean을 초기화하자. 의뢰기로부터 전달된 actionTag와 JSP를 사용하여 id와 str을 초기화하였다.

<jsp:useBean id= "cs2" class= "chap5.ConscriptletTest">

<jsp:setProperty name= "cs2" property= "*" />

</jsp:useBean>

jsp:setProperty의 property에 《*》을 사용하면 의뢰기로부터 전송되여온 자료의 이름을 검사한 후 자동적으로 Bean에 있는 set메쏘드를 사용하여 값을 설정해준다. 만일 일치되는 메쏘드가 존재하지 않아도 오유가 발생하지 않는다.

초기화된 값을 확인하기 위하여 getContent()메쏘드를 사용하였다. 이때 actionTag:JSP가 출력되는것을 확인할수 있다.

<jsp:getProperty name= "cs2" property= "content" />

주의할것은 jsp:useBean액션끄리표를 사용한 초기화는 반드시 파라메터가 없는 구성자를 사용한다는것이다. 만일 ConscriptletTest클라스에 파라메터가 없는 구성자가 존재하지 않는다면 jsp:useBean액션끄리표를 사용할수 없다.

5.12.5. jsp:forward - 다른 페지에 조종권한 넘기기

jsp:forward액션끄리표는 조종권한을 가진 페지가 다른 페지를 호출하면서 자기가 가진 모든 권한을 그 페지로 넘기는 역할을 한다. 이 끄리표는 사용법만 조금 차이날뿐 pageContext의 forward메쏘드와 같은 기능을 가진다. 그러나 XML끄리표형식으로 되여있으므로 좀 더 기능이 추가되여있다.

jsp:forward액션끄리표의 사용형식을 보자. 사용형식에는 여러가지가 있는데 중요한것은 조종권한을 넘길 페지의 경로를 지정하여야 한다는것이다.

<jsp:forward page="path"/>

그러면 jsp:forward액션끄리표의 실례를 보기로 하자.

forward1.jsp는 처음 현시되는 페지이고 forward2.jsp는 조종권한을 넘겨받는 페지이다.

실례 5-22

forward1.jsp
<%@ page contentType= "text/html;charset = big5"%> 처음 시작 페지 이다. <H1>forward1.jsp의 Head</H1> <%

```
    String str = request.getParameter("name");
    out.println(str);
%>
<jsp:forward page="forward2.jsp"/>
 <H1>forward1.jsp의 Tail</H1>
```

forward2.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
forward2.jsp의 출력 : 나중에 만들어진 페지이다. 조종권한이 넘어 왔습니다.
<%=request.getParameter("name")%>
<H1>forward2.jsp</H1>
```

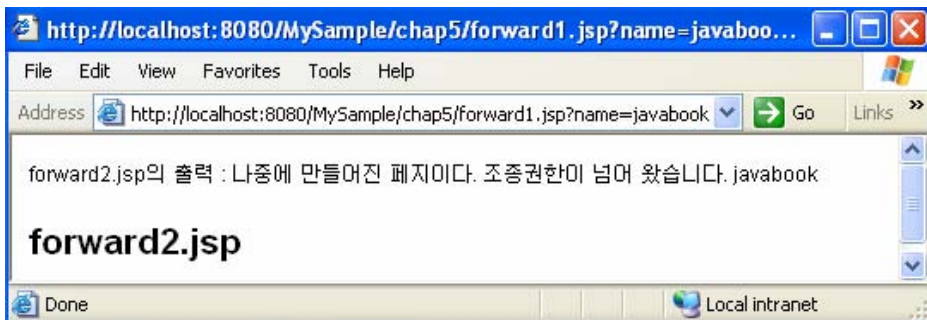



그림 5-28. 실례 5-22의 실행결과

프로그람설명

먼저 forward1.jsp파일을 실행한다. 그러나 결과화면을 보면 forward2.jsp페지의 내용만이 출력되였다(그림 5-28). 그 리유는 다음과 같다.

우선 forward1.jsp페지를 실행시키고 페지의 내용들을 완충기에 저장한다. 이렇게 순서대로 실행하다가 jsp:forward액션끄리표를 만나면 끄리표가 지정하는 페지로 조종권한을 넘긴다. 이때 완충기에 읽어 들였던 내용들은 지워진다. 즉 out.clearbuffer()메쏘드가 호출된 다음 조종권한이 넘어간다. 조종권한을 얻은 새로운 페지는 처음부터 완충기에 새로운 내용들을 읽어 들이고 출력을 진행한다. 출력을 하는 순간 모든 역할은 끝나게 된다. 즉 forward1.jsp는 시작만 하고 아무 내용도 출력하지 못하는 결과를 초래한다. 이것이 jsp:forward액션끄리표의 역할이다.

```
<jsp:forward page="forward2.jsp"/>
```

모든 권한을 forward2.jsp페지에 넘겨 주고 자기가 가지고있는것은 다 지워버리며 다시는 조종권한을 찾을수 없도록 하는것이 jsp:forward액션끄리표이다. jsp:include액션끄리표는 jsp:forward액션끄리표와는 차이가 있다. 그러면 jsp:include액션끄리표를 보면서 결과를 비교해보자.

5.12.6. jsp:include - 다른 페지에 조종권한 빌려주기

jsp:include액션끄리표는 다른 페지에 조종권한을 빌려주는 역할을 한다. 역시 pageContext의 include()메쏘드와 거의 같은 기능을 수행한다.

형식은 다음과 같다.

```
<jsp:include page="path"/>
<jsp:include page="path"></jsp:include>
```

jsp:forward액션끄리표와 jsp:include액션끄리표의 차이점은 첫째로 다른 페지로 권한이 넘어간 후 다시 자기자신의 페지로 돌아오는가 오지 않는가, 둘째로 다른 페지로 권한이 넘어갈 때 자기가 가지고있던것들을 버리고 가는가 아니면 가져가는가 하는 점이다. 실례를 통해 구체적으로 보기로 하자.

실례 5-23

include1.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
처음 시작 페지 이다.<BR>
<H1>include1.jsp의 Head</H1>
<%
   String str = request.getParameter("name");
   out.println("include1.jsp의 출력:"+str+"<BR><BR>");
%>
<jsp:include page="include2.jsp"/>
 <H1>include1.jsp의 Tail</H1>
```

include2.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
 include2.jsp의 출력 : 나중에 만들어진 페지이다. 조종권한이 넘어 왔습니다.
<%=request.getParameter("name")%>
 <H1>include2.jsp</H1>
```

그림 5-29. 실례 5-23의 실행결과

프로그람설명

원천코드는 앞의 실례와 비슷하나 결과에서는 많은 차이가 있다.

먼저 결과화면의 마지막부분을 보면 《include1.jsp의 Tail》이라고 화면에 현시된것을 볼수 있다. 이것은 include2.jsp로 조종권한이 넘어간 후 다시 include1.jsp에게 조종권한이 넘어왔다는것을 보여준다. 또한 결과화면의 처음 부분을 보면 jsp:include액션꼬리표를 실행하기 전까지 include1.jsp에 있던 내용들이 모두 출력된것을 볼수 있다.(그림 5-29)

5.12.7. jsp:forward - 파라메터 포함하기

앞에서 jsp:forward와 jsp:include에 대하여 고찰하였다. 앞의 실례에서는 페지를 호출할 때 URL에 name이라는 변수값을 주어 이 값을 가지고 조종권한이 넘어갔다. 이때 jsp:param꼬리표를 사용하여 변수값을 지정할수도 있다. 그러면 조종권한을 넘길 때 param꼬리표를 사용하여 파라메터값을 넘기는 방법에 대하여 보자.

jsp:param꼬리표는 아래와 같은 형식으로 사용된다.

```
<jsp:forward page="path">
<jsp:param name= value=/>
</jsp:forward>
```

실례 5-24

forward3.jsp?name=love&test=good

```
<%@ page contentType= "text/html;charset = big5"%>
처음 시작 페지이다.<BR>
<H1>forward3.jsp의 Head</H1>
<jsp:forward page="forward4.jsp">
    <jsp:param name = "name" value="javabook"/>
</jsp:forward>
<H1>forward3.jsp의 Tail</H1>
```

forward4.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
forward4.jsp의 출력 : 나중에 만들어진 페지이다. <BR>
조종권한이 넘어 왔습니다. <BR>
<%=request.getParameter("name")%><BR>
<%=request.getParameter("test")%>
<H1>forward4.jsp</H1>
```

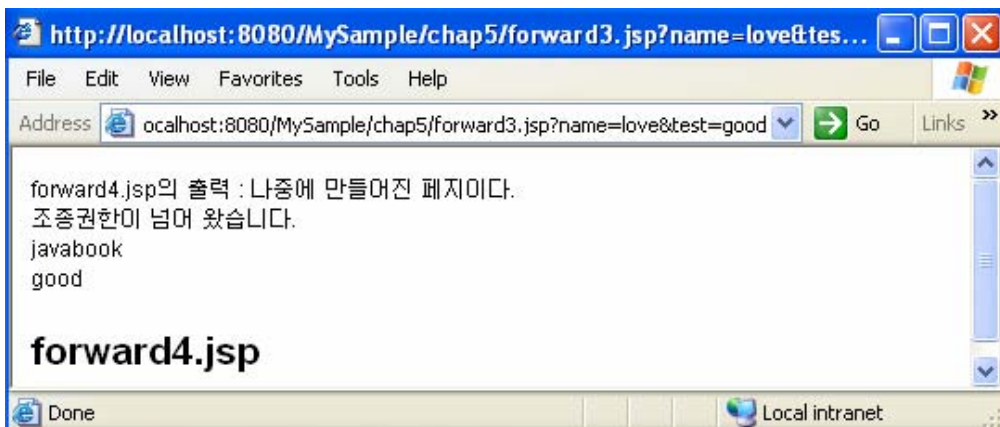



그림 5-30. 실례 5-24의 실행결과

프로그람설명

이 실레는 forward1.jsp실례를 약간 변경한 후 URL에 name=love&test=good라는 값을 주고 실행시킨것이다.

http://localhost:8080/MySample/chap5/forward3.jsp?name=love&test=good

실례를 보면 URL에서 name과 test의 값을 직접 넣어주고있다. 그러나 jsp:forward를 실행할 때 jsp:param끄리표를 사용하여 name에 들어갈 값을 javabook로 바꾸어 이 값이 현시되였다. 여기에서 jsp:forward로 값을 넘길 때 jsp:param끄리표로 넘기는 값이 우선시된다는것을 알수 있다. 이것은 jsp:include끄리표에서도 마찬가지이다.

5.12.8. jsp:include - 파라메터 포함하기

jsp:forward끄리표와 마찬가지로 jsp:include끄리표에서도 같은 형식을 가진다.

```
<jsp:include page="path">
<jsp:param name=value=/>
</jsp:include>
```

URL과 jsp:param끄리표로 동시에 변수값을 넘겼을 때 조종권한이 돌아온 후 어떻게 되는가를 실례를 통하여 고찰하자. 실행결과는 그림 5-31에서 보여주었다.

실례 5-25

include3.jsp?name=love&test=good

```
<%@ page contentType= "text/html;charset = big5"%>
 처음 시작 페지 이다.<BR>
<HTML>
<BODY>
<H1>include3.jsp의 Head</H1>
<%
    out.println(request.getParameter("name")+"<BR>");
    out.println(request.getParameter("test")+"<BR>");
%>
<jsp:include page="include4.jsp">
    <jsp:param name = "name" value="jabook"/>
</jsp:include>
<%=" 조종권한이 돌아 왔습니다. : <BR>"%>
<%= request.getParameter("name")+"<BR>"%>
<%= request.getParameter("test")+"<BR>"%>
<H1>include3.jsp의 Tail</H1>
</BODY>
</HTML>
```

include4.jsp

```
<%@ page contentType= "text/html;charset = big5"%>
 <BR>include4.jsp의 출력 : 나중에 만들어진 페지이다. <BR>
조종권한이 넘어 왔습니다. <BR>
<%=request.getParameter("name")%><BR>
<%=request.getParameter("test")%>
<H1>forward4.jsp</H1>
```

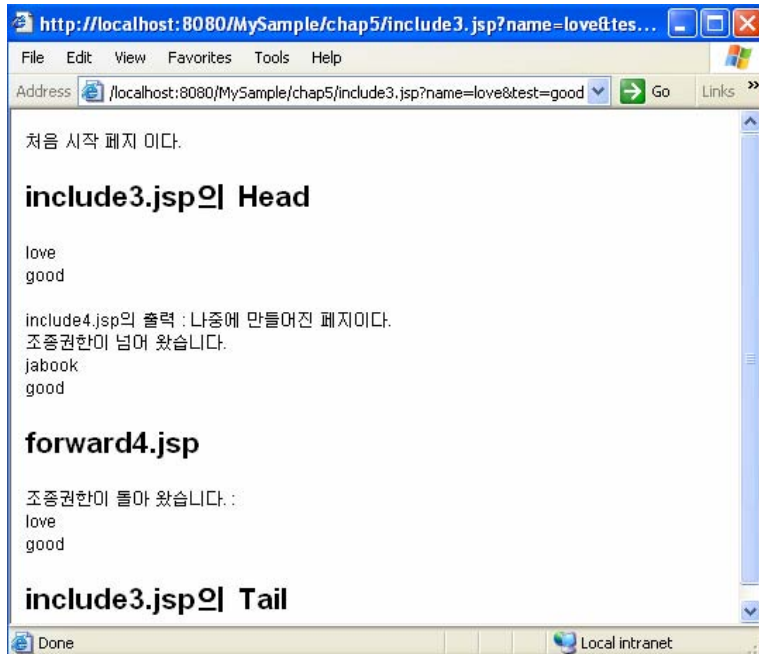



그림 5-31. 실례 5-25의 실행결과

프로그람설명

사용형식은 jsp:forward끄리표와 같다.

<jsp:include page="include4.jsp">

<jsp:param name = "name" value="jabook"/>

</jsp:include>

결과에서 include4.jsp페지로 조종권한이 넘어갈 때 jabook라는 값이 넘어가는것을 확인할수 있다. 그러나 조종권한이 돌아온 후 출력되는 값에 주목을 돌려야 할 점이 있다.

<%=" 조종권한이 돌아 왔습니다. :
"%>

<%= request.getParameter("name")+"<BR>"%>

<%= request.getParameter("test")+"<BR>"%>

jsp:include끄리표를 리용하면 jsp:forward끄리표와는 달리 조종권한이 돌아온 후 원래 값이 다시 호출된다. 즉 이 실례에서는 조종권한이 돌아온 후 love라는 원래값이 다시 호출되였다.

여기에서 알수 있는것은 jsp:param끄리표의 내용은 조종권한을 받은 페지에서만 적용된다는것이다. 그러므로 다시 조종권한이 원래 페지로 넘어왔을 경우 jsp:param끄리표로 넘긴 값들은 사용되지 않는다.

5.12.9. jsp:plugin – jsp파일에 Applet포함하기

보통 Servlet 등 봉사기측에서 제공하는 부품을 실행하면 봉사기측의 자원을 사용하므로 봉사기측에 부하를 주게 된다. 이때 의뢰기가 내리적재받아서 실행한다면 의뢰기의 자원을 사용하게 되므로 봉사기의 부하를 줄일수 있다. 그러나 웨브열람기에 탑재된 Java가상기계는 대체적으로 Java1.0 등의 낡은 판본에 따르므로 최신 판본의 JavaSDK를 리용하여 Applet프로그람을 만들었으면 의뢰기의 웨브열람기에서는 원하는 결과가 나오지 않을수 있다.

이를 위하여 Sun회사에서는 웨브열람기를 위한 최신 판본의 실행시간을 끼워넣기형식으로 제공하고있다.

– 최신판본 JRE(Java Runtime Envirnment)에서 Applet사용의 우점

- 봉사기측의 부하를 덜어준다.
- 최신 Java기술을 사용할수 있다.(Swing, 2DGraphics)
- Applet의 실행속도를 높일수 있다.

이렇게 끼워넣기로 실행할 Applet를 만들었다면 일반적으로 Applet를 실행시킬 때 사용하는 applet꼬리표대신에 object꼬리표(Internet Explorer에서 사용)나 embeded꼬리표(Netscape에서 사용)를 사용한다. 그 리유는 applet꼬리표를 사용하면 웨브열람기는 자신이 기본적으로 탑재한 낡은 판본의 실행시간(run time)을 사용하여 프로그람을 실행시키기때문이다. 그러나 object꼬리표나 embeded꼬리표를 사용하는것은 상당히 복잡하고 까다롭다. 그래서 JSP에서는 Applet나 다른 Java Bean파일 등을 의뢰기에서 끼워넣기형식으로 동작시키려고할 때 자동적으로 열람기의 종류를 확인하고 HTML꼬리표까지 생성시켜주는 jsp:plugin액션꼬리표를 제공한다.

– JRE Plugin에서 Applet의 사용방법

- Ms Explorer: Object꼬리표를 리용하여 복잡한 사용법을 지정한다.
- Nescape : Embed꼬리표를 리용하여 복잡한 사용법을 지정한다.
- jsp:plugin 꼬리표 : Plugin사용을 위한 HTML꼬리표를 자동적으로 만들어준다.

jsp:plugin액션꼬리표를 사용하려면 몇가지 속성을 지정해야 한다.

보통 HTML에서 Applet를 사용할 때 지정하는것과 같은 type, code, width, height 속성들을 지정해주어야 한다. 그외에 몇가지 속성이 더 있는데 표 5-11에서 서술하였다.

표 5-11. jsp:plugin속성들

Jsp:plugin속성들	
type	끼워넣기될 부품류형으로서 applet나 bean이 될수 있다.
code	Applet나 Bean의 클라스명을 지정한다.
width	Applet 등이 표시될 령역의 넓이를 화소단위로 표시한다.
height	Applet 등이 표시될 령역의 높이를 화소단위로 표시한다.
codebase	Applet 등이 위치한 등록부를 지정한다.
align	Applet 등의 맞추기방식을 지정한다.
archive	관련된 클라스파일을 묶어놓은 archive파일을 지정한다.
hspace	Applet 등의 수평여백을 화소단위로 표시한다.
vspace	Applet 등의 수직여백을 화소단위로 표시한다.
name	Java스크립트 등의 스크립트언어에서 Applet를 구별할 이름을 지정한다.
jreversion	Applet 등의 실행에 필요한 JRE의 판본을 지정한다.
nspluginurl	Netscape용 Java끼워넣기를 내리적재할 URL을 지정한다.
iepluginurl	Internet Explorer용 Java끼워넣기를 내리적재할 URL을 지정한다.

그러면 jsp:plugin꼬리표를 리용하여 Applet를 적재하는 간단한 JSP를 만들어보자. 여기서는 JDK에서 견본프로그람으로 제공하는 시계(Clock)Applet를 사용하였다.

실례 5-26

usePlugin.jsp

```
<jsp:plugin type="applet" code="Clock.class" width="300" height="300" >
</jsp:plugin>
```

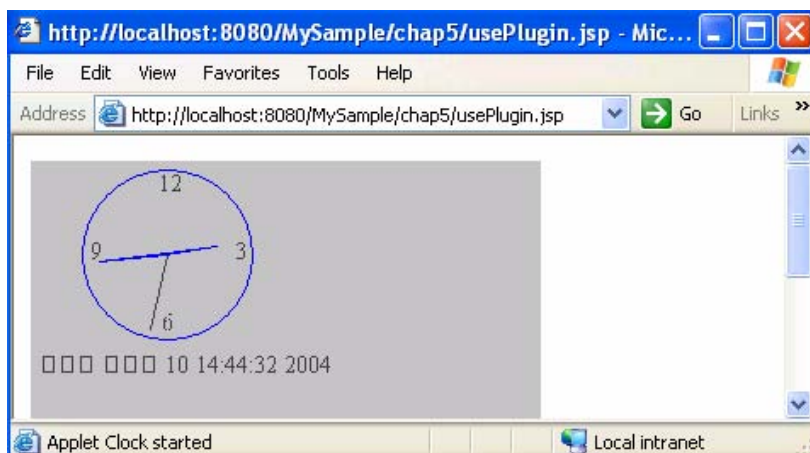
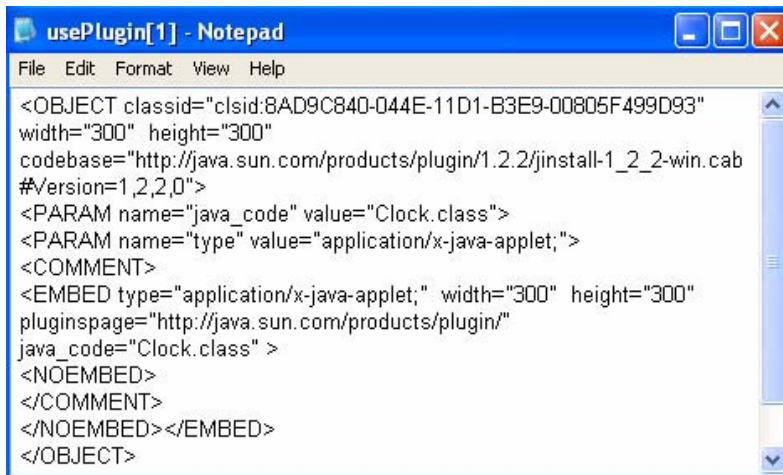



그림 5-32. 실례 5-26의 실행결과

프로그람설명

프로그람을 실행시키면 Applet가 열람기에 현시되게 되며 열람기의 Veiw차림표의 Source항목을 선택하여 HTML의 내용을 확인할수 있다.(그림 5-32) 그 내용을 보면 JSP에서는 Internet Explorer와 Netscape를 위한 Object끄리표와 Embed끄리표를 동시에 생성하고 있다. 그러나 Internet Explorer에 해당하는 Object끄리표부분만 실행되고 Netscape용의 Embed끄리표에 해당한 부분은 실행되지 않는다는것을 알수 있다.

이렇게 jsp:plugin끄리표를 사용하면 최신판본의 JRE를 사용할수 있으므로 Applet와 Bean파일을 보다 빨리 적재할수 있고 낡은 판본의 열람기에서도 2DGraphics, Swing 등 Java의 고급도형대면을 활용할수 있다.

5.12.10. jsp:param-plugin부품에 파라메터값 지정하기

Applet 등의 코드를 실행할 때 참조할 인수들의 이름과 값을 설정하기 위하여서는 jsp:param끄리표를 리용하여야 한다. jsp:param끄리표는 jsp:plugin끄리표의 내부에서 쓰이며 그 형식은 Applet의 param과 비슷하다. jsp에서 사용하는 액션끄리표들은 모두 XML 명령문에 따르므로 대소문자의 구별, 끄리표의 형식 등에 주의하여 사용하여야 한다.

jsp:param의 형식은 다음과 같다.

<jsp:param name = “paramName” value = “paramValue” />

여기서 name은 파라메터이름이고 value는 그의 값이다. 이렇게 지정하면 이것에 대응하는 Applet코드에서 getParameter(“name”)의 형식으로 value값을 참조할수 있다.

jsp:plugin과 jsp:param끄리표를 리용하여 JSP에서 Applet로 값을 넘기는 간단한 실례를 보기로 하자. 실행결과는 그림 5-33에서 보여주었다.

실례 5-27

ParamApplet.java

```
import java.applet.*;
import java.awt.*;
public class ParamApplet extends Applet {
  String paramValue;
  public void start(){
    this.paramValue = getParameter("paramval");
  }
  public void paint(Graphics g){
    g.drawString(paramValue, 50, 50);
  }
}
```

ParamApplet.jsp

```
 <jsp:plugin type="applet" code="ParamApplet.class"
        width="300" height="300" >
          <jsp:params>
                  <jsp:param name="paramval" value="Hello!! jsp:param" />
          </jsp:params>
</jsp:plugin>
```

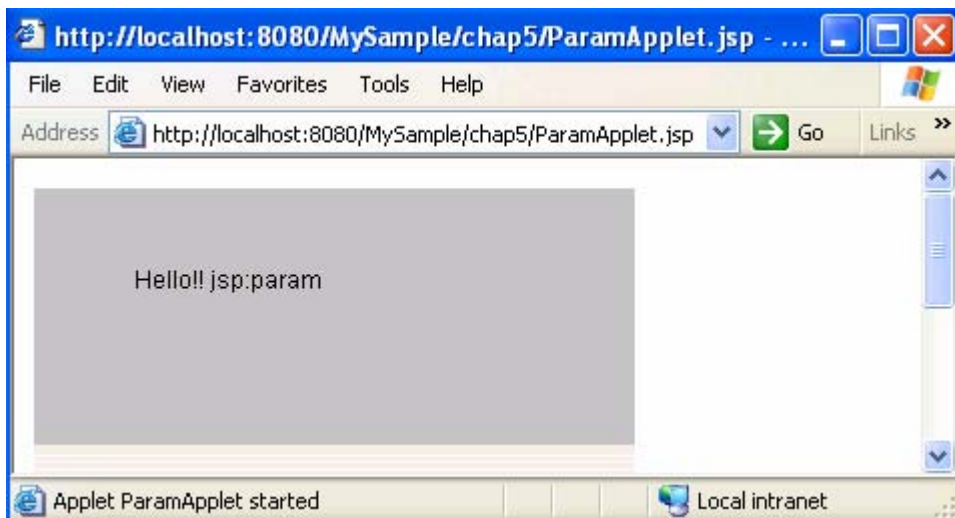



그림 5-33. 실례 5-27의 실행결과

프로그람설명

ParamApplet.jsp파일을 보면 그 형식이 철저히 XML의 형식에 따르고있다는것을 알수 있다.

```
<jsp:plugin type="applet" code="ParamApplet.class" width="300" height="300" >
<jsp:params>
<jsp:param name="paramval" value="Hello!! jsp:param" />
</jsp:params>
</jsp:plugin>
```

여기서 주의를 돌려야 할 점은 <jsp:params>끄리표의 아래에 다시 <jsp:param>끄리표가 쓰이였다는것이다. 이것은 파라메터값이 여러 개가 될수 있다는것을 의미하며 <jsp:params>끄리표의 아래에는 여러 개의 <jsp:param>끄리표가 들어갈수 있다. 이 실례에서는 하나의 파라메터만을 지정하고있는데 그 내용을 보면 paramval이라는 인수에 "Hello!! jsp:param"이라는 문자렬을 값으로 지정하고있다.

그리고 ParamApplet.java에서는 이 값을 받아 리용한다.

```
this.paramValue = getParameter("paramval");
```

이렇게 JSP에서 Applet로 파라메터를 넘기는 방법에 대하여 보았다. 이것을 JavaScript 등을 리용하여 페지를 만든다면 보다 동적인 페지를 만들수 있다.

제6장. 대화접속과 쿠키

제1절. 대화접속

《대화접속(session)》은 봉사기가 의뢰기측의 망열람사용자정보를 개별처리하고 기록하는 기술이다. 거점을 열람할 때 사용자는 열람기프로그람을 가지고있기만 하면 거점에서 보호되지 않은 홈페지들을 마음대로 열수 있다. 이 문제를 해결하기 위하여 나온것이 대화접속이다.

이 절에서는 JSP의 《session》객체를 리용하여 《대화접속》기능을 실현하는 방법을 서술한다.

6.1.1. 대화접속의 개념과 사용방법

대화접속은 봉사기가 망열람하는 매 사용자에게 한개의 "대화접속"을 정하고 그의 내용을 제공함으로써 봉사기측이 대화접속을 통해 사용자정보를 식별하고 개별적인 봉사를 제공하도록 하는 기능을 수행한다.

이 대화접속들은 개별적으로 독립이고 자기의 존재시간을 가진다. 매 망열람사용자들에 대한 대화접속은 봉사기측프로그람에서 만들어지며 사용자가 봉사기에 요청을 하여 제정된 한도시간에까지 접근하지 못하면 봉사기에서 자동적으로 지워진다. 일반적으로 Tomcat에서 대화접속이 존재하는 시간은 30분이다. 아래에서 대화접속을 어떻게 만들며 어떤 설정들을 할수 있는가를 고찰한다.

1) 사용자대화접속의 작성

JSP에서 사용자의 대화접속을 작성하려면 반드시 request객체의 메쏘드를 사용하여야 한다. 이 형식은 다음과 같다.

Request.getSession(true)

2) 대화접속에서 자료의 추가와 삭제

사용자대화접속을 만든 다음 대화접속에서 자료를 추가하고 삭제하려면 반드시 session객체를 사용하여야 하며 이 객체는 《javax.servlet.http.HttpSession》에 의하여 파생된다. 아래에서는 자료를 추가하고 삭제하는 메쏘드들에 대하여 설명하였다.(표 6-1)

표 6-1. 대화접속 메쏘드들(1)

메 쏘 드	설 명
putValue("문자렬변수", "자료")	대화접속에서 한개의 문자렬변수를 설정하고 그것에 자료내용을 설정한다.
removeValue("문자렬변수")	대화접속에서 지정한 문자렬변수와 그 내용을 삭제한다.

3) 대화접속에서 변수내용얻기

아래의 메쏘드들은 대화접속으로부터 그 안에 보존한 매 정보들을 얻는 메쏘드이다.(표 6-2)

표 6-2. 대화접속 메쏘드들(2)

메 쏘 드	설 명
getValue(“문자렬변수”)	대화접속에서 지정한 문자렬변수의 내용을 얻는다.
getValueNames()	대화접속에서 모든 문자렬변수의 이름들을 추출하여 문자배렬로 귀환한다.

아래의 실례는 사용자가 홈페지를 연 다음 대화접속을 만들고 새 변수 《user》와 《ip》에 각각 내용을 설정한 다음 대화접속의 자료를 추출하여 화면상에 현시하는 프로그람이다.

실례 6-1

ch6-1.jsp(session의 작성)

```
<HTML>
<TITLE> session의 작성</TITLE>
<BODY>
<%
  request.getSession(true); //session의 작성
  session.putValue("user","김영철");
   //변수 user를 설정하고 session에 넣는다.
  session.putValue("ip",request.getRemoteAddr());
  /* 변수 ip를 사용자의 망열람 IP로 설정하고 session에 넣는다. */
  out.print("<font color=blue size=5>");
  out.print(session.getValue("user")+"</font> 당신은 ");
  /* session에서 user변수의 내용을 현시한다.     */
  out.print("<font color=red size=5>");
  out.print(session.getValue("ip")+"</font>에서 옵니다.<p>");
  /* session에서 ip변수의 내용을 현시한다 .*/
  String a[]=session.getValueNames(); //session에서의 모든 변수들을 얻는다.
  out.print("session에서의 변수들: ");
  for(int i=0;i<a.length;i++)
  out.print("("+a[i]+")");
%>
</BODY>
</HTML>
```

일단 사용자대화접속을 만들고 이것이 유효시간을 초과하면 체계에 의해 그것이 지워져야 한다. 그렇지 않으면 봉사기측의 임의의 페지에서 여기 자료들을 읽어 들일수 있다. 이 실례의 결과를 그림 6-1에서 보여주었다.

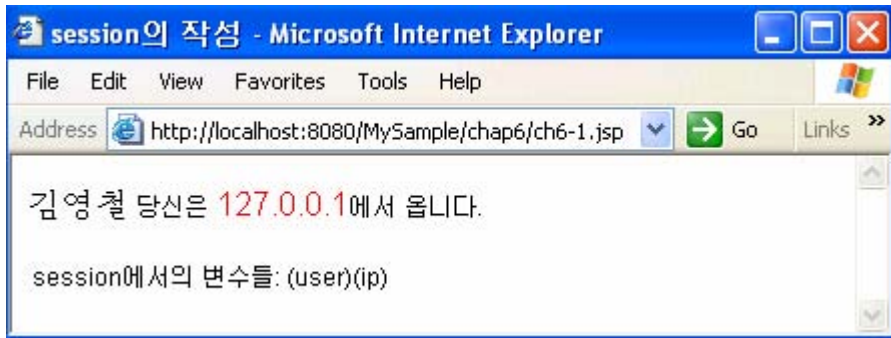



그림 6-1. 실례 6-1의 실행결과

4) session객체의 메쏘드

대화접속의 작성 및 자료를 대화접속에 새로 추가하는 메쏘드 외에 session객체는 몇가지 메쏘드들을 더 가지고있다. 이 메쏘드들에 대하여 표 6-3에서 보여주었다.

표 6-3. 대화접속 메쏘드들(3)

메 쏘 드	설 명
getID()	대화접속의 봉사기측식별코드를 귀환한다.
isNew()	현재 사용자의 대화접속이 새로운 대화접속인가 아닌가를 판단한다. 만일 새로운것이면 "True"를 귀환하고 아니면 "False"를 귀환한다.
invalidate()	현재 사용자의 대화접속을 삭제한다.
getCreationTime()	대화접속존재시간을 귀환한다. 귀환값의 단위는 ms이다.
getLastAccessedTime()	마지막 사용자가 대화접속을 통하여 봉사기에 요구를 하는 시간을 귀환한다. 귀환값의 단위는 ms이다

다음으로 《getID》와 《isNew()》메쏘드에 대하여 구체적으로 설명한다.

(1) 사용자의 대화접속을 작성할 때 이것을 식별할수 있는 하나의 식별코드를 만든다. 봉사기는 식별코드에 따라 대화접속을 구별하고 정확히 처리하며 봉사를 제공한다.

즉 《session.getID()》명령은 《To1019mC3207364782061355At》와 같은 식별코드를 귀환한다.

(2) 《isNew()》메쏘드는 봉사기측이 이미 현재의 사용자에 대하여 대화접속을 작성하고 있는가를 판단한다. 실례로 실례 6-1의 마지막에 아래의 코드

```
out.print(session.isNew());
```

를 추가하면 대화접속이 새로 작성되지 않은 경우에 열람기상에 《false》가 현시된다.

5) 대화접속의 존재시간

프로그람을 실행하여 의뢰기측사용자에 대한 대화접속을 만든 다음 의뢰기가 봉사기에 아무런 요청을 하지 못한 상태에서 존재시간을 초과하면 대화접속은 자동적으로 결속된다. 또한 사용자가 열람기를 직접 닫거나 프로그람에서 《invalidate()》메쏘드를 사용하여 강제적으로 대화접속을 끝낼수도 있다.

session객체에는 대화접속의 존재시간을 얻고 설정하는데 리용하는 2개의 메쏘드가 있다.(표 6-4)

표 6-4. 대화접속 메쏘드들(4)

메 쏘 드	설 명
getMaxInactiveInterval()	기정으로 설정된 대화접속의 존재시간을 얻는다. 단위는 초이다.
setMaxInactiveInterval(설정시간)	대화접속의 존재시간을 새로 설정한다. 이것은 사용자가 봉사기에 요청을 하지 않은 상태에서 대화접속이 자동적으로 결속되는 시간간격을 의미한다. 단위는 초이다.

이 2가지 메쏘드를 사용한 실례를 보기로 하자.

실례 6-2

ch6-2.jsp(session의 존재시간을 현시하고 수정)

```
<HTML>
<TITLE>session의 존재시간을 현시 및 변경</TITLE>
<BODY>
<%
   out.print("session의 기정존재시간:");
   /* 대화접속의 기정존재시간을 얻고 현시 */
   session.setMaxInactiveInterval(3600);
   /* 새로운 대화접속의 존재시간을 설정 */
   out.print("session설정후의 존재시간: ");
   out.print(session.getMaxInactiveInterval()+" 초<BR>");
%>
</BODY>
</HTML>
```

프로그람설명

이 실례에서는 먼저 session.getMaxInactiveInterval()메쏘드를 사용하여 대화접속의 기정존재시간을 얻는다.

또한 《session.setMaxInactiveInterval(3600);》를 리용하여 대화접속의 존재시간을 3600s로 다시 설정한다.

6.1.2. 대화접속의 응용

여기에서는 대화접속을 리용한 여러가지 구체적인 실례에 대하여 보기로 한다.

1) 간단한 계수기

망열람인원수의 계수작업은 session객체를 리용하면 아주 쉽게 할수 있다. 그 방법은 다음과 같다.

① 망열람하는 사용자수를 얻기 위하여 열람기상의 refresh단추를 찰칵하여 망열람자수를 증가시킨다.

```
<%! int count=0;%>
<%
count++;
out.print( "당신은 이 거점의" + count+ "번째 망열람자입니다");
%>
```

프로그람에서 계수변수 count의 초기값을 0으로 설정한다. JSP프로그람이 해석된 후 servlet로 되여 의뢰기측에 존재하므로 그 다음 사용자의 계속적인 요청에 이 servlet에 의하여 응답하게 된다. 망열람하는 사용자가 있을 때마다 count변수의 값을 1씩 증가시킨다. 이때 이 값은 servlet에 놓여있기때문에 봉사기를 닫기 전까지는 그 값이 잃어지지 않는다.

② 앞의 설명으로부터 알수 있는바와 같이 모든 망열람사용자들은 servlet의 count변수를 공유하고있다. 이로 인하여 하나의 새로운 문제가 발생될수 있다. 례를 들면 망열람하는 경우 자신이 첫번째 망열람자이고 또 망열람하는 다른 사용자가 있다면 이때 servlet의 count값은 변경된다. 만일 이때 refresh단추를 찰칵하면 다른 망열람자로 바뀌여질수 있다. 이렇게 되면 이상한 감을 느낄수 있다.

이 문제를 해결하기 위하여 여기서는 한가지 기법을 사용하고있다. 즉 사용자가 망열람할 때의 count값을 다른 변수 usercount에 넣어 망열람자의 망열람순서를 현시할 때 usercount의 내용을 현시한다. usercount는 사용자대화접속에 존재하는 하나의 정보이며 사용자가 열람기를 닫거나 기정시간을 연장하여 넘기지 않고서는 변경되여도 잃을수 없다. 아래에서 완전한 프로그람을 보여준다.

실례 6-3

ch6-3.jsp (간단한 계수기)

```
<HTML>
<TITLE>간단한 계수기</TITLE>
<BODY>
<%! int count=0; %>
<%
String usercount; //usercount는 망열람인원수를 기록
request.getSession(true); //사용자의 대화접속을 작성
if(session.isNew()) //현재 사용자에 대해 이미 대화접속을 작성한적이 있는가를 판단
{
        count++;
        usercount=String.valueOf(count);
        session.putValue("usercount",usercount);
}
out.print("당신은 이 거점의 <font color=red size=5>");
out.print(session.getValue("usercount"));
out.print("</font>번째 망열람자입니다");
%>
</BODY>
</HTML>
```

프로그람설명

여기서는 사용자가 망열람할 때마다 《request.getSession(true)》로 사용자의 대화접속을 만든다. 다음 《if(session.isNew())》를 리용하여 새로운 대화접속인가 아닌가를 판단하며 만일 새로운 대화접속이라면 《count》값을 하나 증가시키고 사용자대화접속의 《usercount》변수에 넣는다. 《usercount》변수를 리용하여 방금 접속한 사용자가 몇번째로 망열람하는 망열람자인가를 현시한다.

만일 사용자가 열람기상의 refresh단추를 찰칵하면 프로그람은 《if(session.isNew())》조건명령문아래의 프로그람블로크를 집행할수 없다. 왜냐하면 그 사용자는 이미 자기의 대화접속을 만들어 놓은 상태이기때문이다. 이것은 refresh단추를 찰칵한다고 하여 대화접속이 변경되는것이 아니라는것을 의미한다. 열람기상의 다음에 현시하는 망열람순서는 대화접속의 《usercount》변수의 내용으로서 그림 6-2와 같다.

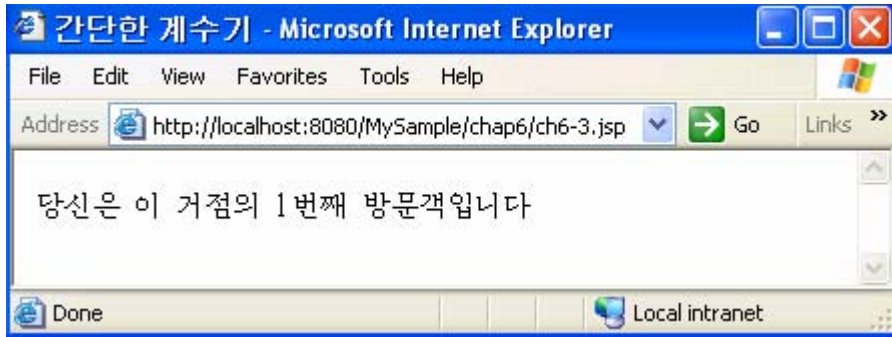



그림 6-2. 실례 6-3의 실행결과

2) 사용자의 주페지에 들어가기

일반적으로 거점은 많은 홈페지와 련결할수 있으며 이 홈페지들이 특별한 보호가 되여있지 않다면 사용자는 열람기상의 주소렬에 어느 한 페지의 망주소를 입력하여 열람할수 있다. 주페지에서 사용자정보보다 가입자인증을 요구하는 거점들에서는 이것이 허용될수 없다. 이러한 문제점은 대화접속을 리용하여 해결할수 있다. 아래에서 사용자의 주페지에 들어가는 실례에 대하여 보여주었다.

실례 6-4

ch6-4.jsp (사용자의 주페지에 들어가기)

```
<HTML>
<TITLE>거점의 첫페지</TITLE>
<BODY>
<%
  request.getSession(true); //대화접속작성
  session.putValue("enter","true"); //enter변수를 true로 설정
%>
<center>
<font color=green size=5><b> 안녕하십니까, 이 거점에 오신것을 환영합니다!
</b></font>
<a href=ch6-5.jsp>최근정보열람</a>
<img src=zct.jpg>
<font color=red size=4>
<%
   String msg=request.getParameter("msg");
   if(msg != null)
   out.print("<BR>"+msg);
%>
</font>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 거점의 주페지이며 첫번째 <% %>사이의 코드는 대화접속을 만들고 enter항목을 true로 설정하였다. 결과 사용자가 기타 홈페지를 열람할수 있는 통행증을 얻은것과 같게 되였다.

두번째 <% %>사이의 코드는 다음 페지(ch6-5.jsp)가 귀환하는 오유정보를 현시하도록 한다.

실례 6-5

ch6-5.jsp (사용자의 주페지에 들어가기(계속))

```
<HTML>
<TITLE>거점의 내부페지</TITLE>
<BODY>
<%
   if((String)session.getValue("enter")!="true")
   response.sendRedirect("ch6-4.jsp?msg=먼저 주페지의 정보를 보십시오!");
%>
<img src=news.jpg>
</BODY>
</HTML>
```

프로그람설명

이 홈페지는 거점의 다른 련결페지로서 사용자가 이 페지를 열 때 우선 대화접속의 《enter》항목이 《true》와 같은가 같지 않는가를 검사하며 같지 않으면 사용자가 정상방식으로 이 페지를 열수 없다. 그리고 《response.sendRedirect(“ch6-4.jsp?msg=먼저 주페지의 정보를 보십시오!!);》은 첫페지로 강제로 돌아와 오유정보를 전달하도록 한다.

아래에서 집행결과를 보여준다.(그림 6-3)

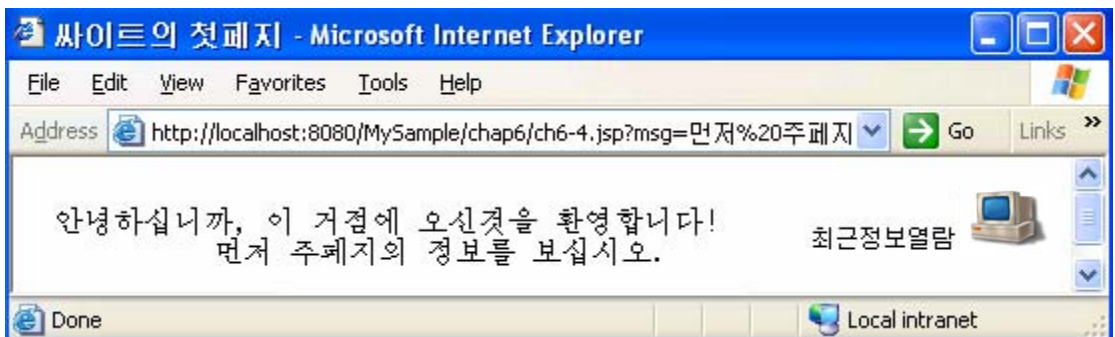



그림 6-3. 실례 6-5의 실행결과

3) 직결식물품구매기능

대화접속은 직결(on-line)로 물품을 구매하는 봉사를 하는데도 쓰인다.

체계는 사용자가 구매하는 상품명과 그 수량을 장악할수 있어야 한다. 매 대화접속은 모두 독립적이므로 매 사용자마다 자기의 《물품구매차》를 가지게 되며 사용자는 직결식상점에 들어가 자기가 좋아하는 상품을 선택할수 있다. 또한 대화접속의 자료는 홈페지의 전환으로 잃어질수 없다. 사용자가 필요한 자료들을 선택한 후 봉사기측프로그람으로 이 대화접속의 자료를 처리하여 직결식물품구매의 목적을 달성한다. 아래에서 대화접속으로 직결식물품구매홈페지를 제작하는 방식을 보여준다.

실례 6-6

ch6-6.jsp(session을 사용한 직결식물품구매홈페지의 제작)

```
<HTML>
<TITLE>session을 사용하여 직결식물품구매페지제작</TITLE>
<BODY>
<center>
<form action=ch6-7.jsp method=Post>
<table border=1>
<tr bgcolor=#C0C0C0><td>상품명 </td>
<td>가격</td><td>수량입력</td></tr>
<tr>
<td bgcolor=yellow>상품 1</td><td>2000</td>
<td><input type=text size=8 name=good1></td>
</tr>
<tr>
<td bgcolor=yellow>상품 2</td><td>1500</td>
<td><input type=text size=8 name=good2></td>
</tr>
<tr>
<td bgcolor=yellow>상품 3</td><td>1000</td>
<td><input type=text size=8 name=good3></td>
</tr>
<tr>
<td bgcolor=yellow>상품 4</td><td>1300</td>
<td><input type=text size=8 name=good4></td>
</tr>
<tr>
<td bgcolor=yellow>상품 1</td><td>1800</td>
<td><input type=text size=8 name=good5></td>
</tr>
```

```
<tr><td colspan=3 align=center><input type=submit value=보내기>
<input type=reset value=다시입력></td>
</table>
</form>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 사용자가 구매상품을 선택하고 구매수량을 입력하는 홈이다. 이 홈을 처리하는 프로그람은 실레 6-7 이며 홈의 매 상품은 하나의 변수이름을 가진다.《good1》~《good5》에는 이 상품을 구매하려는 수량을 기록한다.

실례 6-7

ch6-7.jsp(session을 사용한 직결식물품구매홈페지의 제작(계속))

```
<HTML>
<TITLE>구매상품일람표</TITLE>
<BODY>
<%@page import="java.util.*"%>
<%
String name,count;
request.getSession(true);
Enumeration goods=request.getParameterNames();
while(goods.hasMoreElements())
{
        name=(String)goods.nextElement();
        count=request.getParameter(name);
        session.putValue(name,count);
}
String sname[]=session.getValueNames();
out.print("<font size=5 color=green>구매상품일람표</font><p>");
for(int i=0;i<sname.length;i++)
{
        out.print(sname[i]+"=");
        out.print(session.getValue(sname[i]+"<BR>");
}
%>
</BODY>
</HTML>
```

프로그람설명

우선 사용자에 대한 대화접속을 하나 만들고 우의 홈(실례 6-6)이 전송하는 자료를 처리한다. 《request.getParameterNames()》는 홈의 모든 상품에 대한 변수이름을 얻는다. 귀환하는 자료형은 《Enumeration》이며 이것은 《java.util.*》에 포함되여있다. 때문에 프로그람의 처음에 반드시《<%@page import="java.util.*"%>》를 써주어 java.util.*를 적재해야 한다.

while순환안에서는 변수이름들을 순차적으로 얻어 림시로 변수 《name》에 넣는다. 《request.getParameter(name)》은 변수의 내용을 얻는다.

또한 《session.putValue(name, count)》은 자료를 대화접속에 넣는다.

마지막으로 《session.getValueNames()》은 《sname[]》인 문자배렬을 귀환하며 그 안에는 모든 상품변수의 이름이 들어있다. for순환은 상품변수이름과 그 내용을 현시한다.

이 프로그람의 집행결과를 그림 6-4~6-5에서 보여준다.

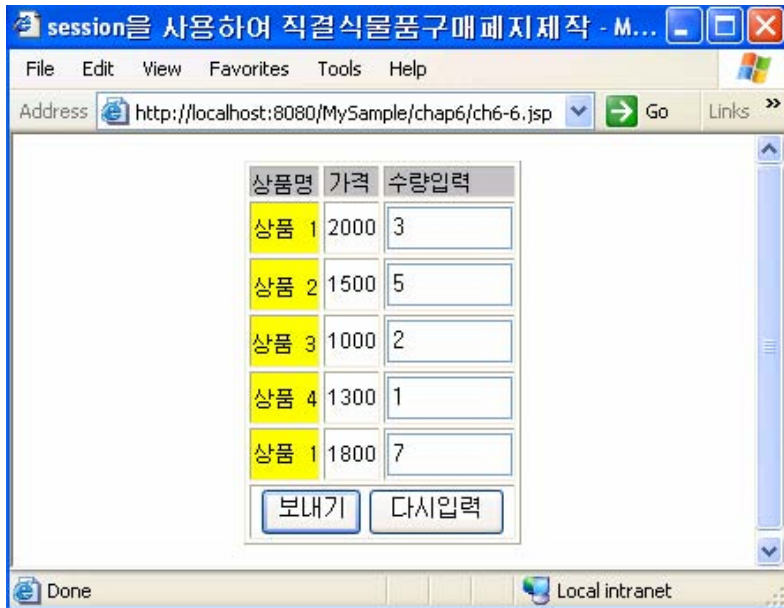



그림 6-4. 실례 6-6의 실행결과

그림 6-5. 실례 6-7의 실행결과

이 프로그람은 단지 사용자가 상품을 구매하는것뿐이다. 사용자가 이 상품구매정보를 보낸 다음에는 봉사기측프로그람이 반드시 총적가격을 계산하거나 자료를 자료기지에 넣어야 한다. 더우기 사용자는 상품구매상점에 들어가기전에 신분확인 등의 작업을 진행하여야 한다.

제2절. 쿠키

의뢰기측에 보관한 사용자정보를 《쿠키》라고 한다. 쿠기를 리용하면 자료의 량적인 측면에서 봉사기측의 처리시간을 감소시킬수 있으며 의뢰기측에 정보가 기록되여있으므로 봉사기 역시 망열람기사용자의 신분을 식별할수 있다. 이밖에 홈페지를 변동시킬 때에 쿠키의 자료도 보존할수 있다. 이 절에서는 쿠키의 개념과 사용방법을 학습한다.

6.2.1. 쿠키의 특징과 기능

쿠키는 열람기가 제공하는 하나의 기술이다. 이 기술은 봉사기측프로그람이 의뢰기측에만 보존할수 있거나 의뢰기측에서 능히 처리를 진행할수 있는 자료를 의뢰기에 넣어줌으로써 꼭 망을 통하여 전달하지 않고도 홈페지의 처리효률을 높일뿐아니라 봉사기의 부하를 감소시킬수 있다. 그러나 쿠키의 사용은 일정한 제한성을 가진다.

1) 쿠키의 작성과 전송

《javax.servlet.http.Cookie》클라스는 쿠키의 처리방식을 제공하고있으며 사용자의 열람기가 쿠키를 요구하기만 하면 아래의 문법에 따라 쿠키를 작성할수 있다.

Cookie 쿠키이름=new Cookie("첨수값", "내용")

쿠키를 작성한 다음에는 HTTP머리부를 통해 응답하고 쿠키를 의뢰기의 콤퓨터상에 보존시킨다. 즉

response.addCookie(쿠키이름)

다음의 프로그람부분은 쿠키를 작성하고 전송하는 실례이다.

```
<%
Cookie user=new Cookie( "id1", "김영철");
response.addCookie(user);
%>
```

이 실레에서는 쿠키이름이 《user》이며 그의 첨수값과 내용은 각각 《id1》과 《김영철》이다. 《response.addCookie(user)》는 작성된 쿠키의 정보를 사용자의 콤퓨터에 보존한다.

2) 쿠키의 존재시간

쿠키의 존재시간은 클라스를 사용하여 쿠키를 만들 때로부터 사용자가 열람기를 닫을 때까지이다. 쿠키의 존재시간을 연장하려면 즉 사용자가 열람기를 닫은 후 여전히 쿠키의 자료를 유지하고 다음번에 사용자가 망열람할 때에 직접 쿠키의 내용을 얻자면 쿠키를 만든 다음 쿠키의 존재시간을 정의하여야 한다. 존재시간을 정의하는 메쏘드는 setMaxAge() 이다. 아래의 실례를 보도록 하자.

```
<%
Cookie user=new Cookie( "id1", "김영철");//쿠키를 작성
user.setMaxAge(3600);// 존재시간을 3600s로 설정
response.addCookie(user);//쿠키를 의뢰기측에 추가
%>
```

첫행 코드에서는 이름이 《user》인 쿠키를 만들고 《user.setMaxAge(3600)》은 쿠키의 존재시간을 개시후 3600s이내로 설정하였다. 즉 이 쿠키의 자료는 한시간동안 보유할수 있다.

3) 쿠키의 존재개수

쿠키의 개수에는 한계가 있다. 한대의 의뢰기측콤퓨터는 최대로 300개의 쿠키를 보존할수 있으며 한 봉사기에 대하여 최대로 20개의 쿠키를 가질수 있다.

4) 쿠키와 대화접속의 차이

앞에서 서술한 대화접속도 사용자개별정보를 보존하는데 쓰인다. 표 6-5에서는 대화접속과 쿠키의 차이를 보여주었다.

표 6-5. 대화접속과 쿠키의 차이

항　목	쿠　키	대화접속
존재시간	열람기를 닫기전 및 설정시간내	열람기를 닫기전 및 기정시간내
존재방식	의뢰기콤퓨터	봉사기측콤퓨터
개수	20개(한 봉사기에 대하여)	제한이 없음. 개수가 많을수록 봉사기효률이 저하
사용객체	Cookie	session
처리속도	빠르다	늦다

자료처리를 망을 통하지 않고 의뢰기측사용자의 콤퓨터에 직접 련결시켜 진행하면 높은 실행효률을 가질수 있고 봉사기측의 부하를 감소시킬수 있다. 처리하여야 할 자료량이 최대개수의 쿠키가 허용하는 범위내에 있으면 쿠키의 기능을 사용하는것이 더 적합하다.

6.2.2. 쿠키의 접근과 사용

여기에서는 쿠키를 어떻게 사용하며 그것을 리용하여 어떤 형태의 홈페지기능을 수행할수 있는가를 고찰한다.

1) 쿠키의 정보얻기

쿠키의 정보를 얻는데는 몇가지 메쏘드들이 있다.(표 6-6)

표 6-6. 쿠키와 관련한 메쏘드들

메쏘드	설　명
request.getCookies()	의뢰기의 쿠키정보를 귀환
쿠키이름.getName()	쿠키의 첨수값을 귀환
쿠키이름.getValue()	쿠키에 저장된 내용을 귀환

아래의 응용프로그람은 우의 3개 메쏘드를 사용하여 의뢰기측쿠키의 정보를 얻는 방법을 보여주고있다.

실례 6-8

ch6-8.jsp(쿠키정보의 얻기)

```
<HTML>
<TITLE>Cookie정보의 얻기</TITLE>
<BODY>
<%
Cookie user1=new Cookie("id1","김영철");
```

```
Cookie user2=new Cookie("id2","김인철");
response.addCookie(user1);
response.addCookie(user2);
Cookie[] allcookie=request.getCookes(); // 모든 쿠키들을 얻는다
%>
<table border=1>
<tr bgcolor=#C0C0C0><td>첨수값</td><td>내용</td></tr>
<%
for(int i=0;i<allcookie.length;i++)
{
    String idx=allcookie[i].getName();//쿠키의 첨수값 얻기
    String value=allcookie[i].getValue();//쿠키의 내용 얻기
    out.print("<tr><td bgcolor=yellow>");
    out.print(idx+"</td><td>");
    out.print(value+"</td></tr>);
}
out.print("</table>");
%>
</BODY>
</HTML>
```

프로그람설명

먼저 2개의 사용자쿠키 《user1》와 《user2》를 만든다. 《Cookie[] allcookie=request.getCookies();》는 모든 쿠키를 얻어 배렬에 넣는다.

다음 for순환은 《getName()》과 《getValue()》메쏘드를 사용하여 쿠키배렬에서 매개 쿠키의 첨수값(idx)과 내용(value)을 얻은 다음 열람기상에 출력한다.(그림 6-6)

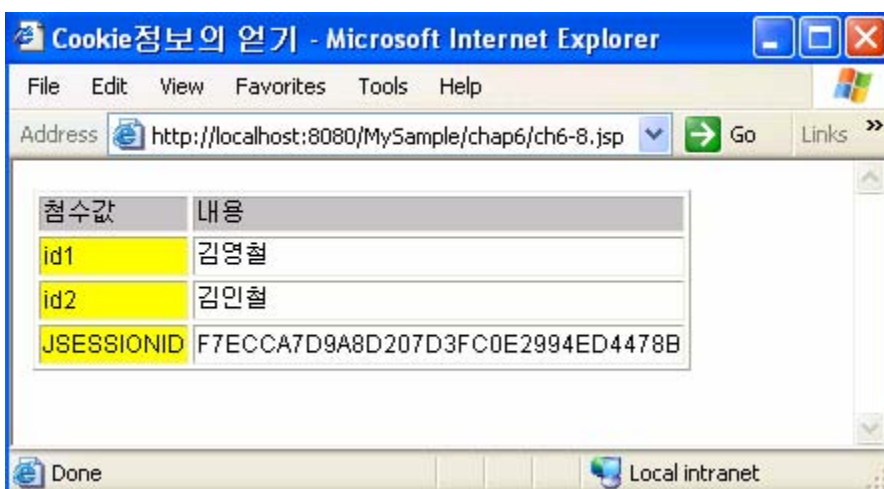



그림 6-6. 실례 6-8의 실행결과

2) 쿠키의 삭제

쿠키를 삭제하려면 쿠키존재시간을 설정하는 메쏘드 《setMaxAge()》를 사용하여 쿠키의 존재시간을 《0》으로 설정하면 된다. 즉

쿠키이름.setMaxAge(0)

3) 쿠키의 응용

쿠키는 자료를 사용자의 콤퓨터에 보존하여 처리를 하므로 실행효률이 가장 높으며 보통 가입자자료를 입력할 때 리용한다.

실례 6-9

ch6-9.jsp(가입자등록거점)

```
<HTML>
<TITLE>가입자등록거점</TITLE>
<BODY>
<font size=5 color=green>등록화면</font>
<font color=red>
<%
  String errmsg=request.getParameter("errmsg");
  if(errmsg!=null)
  out.print(errmsg);
%>
</font>
<hr>
<form action=ch6-10.jsp method=Post>
<table border=0>
<tr><td>이름:</td><td><input type=text size=20 name=name></td>
<td><input type=submit name=send value=등록></td></tr>
</table>
</form>
<a href=ch6-11.jsp>가입자등록페지</a>
</BODY>
</HTML>
```

실례 6-10

ch6-10.jsp(가입자등록거점(계속))

```
<%
String errmsg;
String name=request.getParameter("name");
if(name.length()==0)
{
      errmsg="마당입력은 빌수 없다!";
      response.sendRedirect("ch6-9.jsp?errmsg="+errmsg);
}
else
response.sendRedirect("ch6-11.jsp?name="+name);
%>
```

실례 6-11

ch6-11.jsp(가입자등록거점(계속))

```
<%
   String name=request.getParameter("name");
   if(name==null)
   {
     Cookie[] allcookie=request.getCookies();//모든 쿠키배렬을 얻기
     int count=allcookie.length-1;
     for(int i=0;i<=count;i++)
     {
      String login=allcookie[i].getName();//매 Cookie의 첨수값 얻기
      if(login.length()==4)
      {
        name=allcookie[i].getValue();
        break;
      }
      else if(i==count)
        {
          String errmsg="우선 등록하여야 가입자마당에 들어갈수 있습니다!";
          response.sendRedirect("ch6-9.jsp?errmsg="+errmsg);
        }
     }
   }
   else
```

```
    {
      Cookie login=new Cookie("user",name); //사용자에 대하여 쿠키를 작성
      login.setMaxAge(86400);
      response.addCookie(login);
    }
%>
<HTML>
<TITLE>가입자등록페지</TITLE>
<font color=green size=6>가입자마당</font>
<hr>
<font color=red size=5>
<%=name%>
</font>
, 가입자페지에 온것을 환영합니다
</BODY>
</HTML>
```

프로그람설명

우선 홈페지에 들어가는 사용자가 이미 등록수속을 하였는가를 판단한다. 만일 직접 들어간 사용자이라면 《name=null》로 된다. for순환은 현재 의뢰기의 모든 쿠키들이 합법적인가를 순차적으로 검사한다. 만일 등록되여있으면 이 페지에 들어가고 그렇지 않으면 첫번째 페지에 돌아와 등록한다.

그리고 방금 등록한 사용자이라면 이름이 《login》인 쿠키를 만든다. 다음 코드 《login.setMaxAge(86400);》은 이 쿠키의 존재시간을 개시후부터 86400s로 설정한다. 실행결과는 그림 6-7~6-8과 같다.

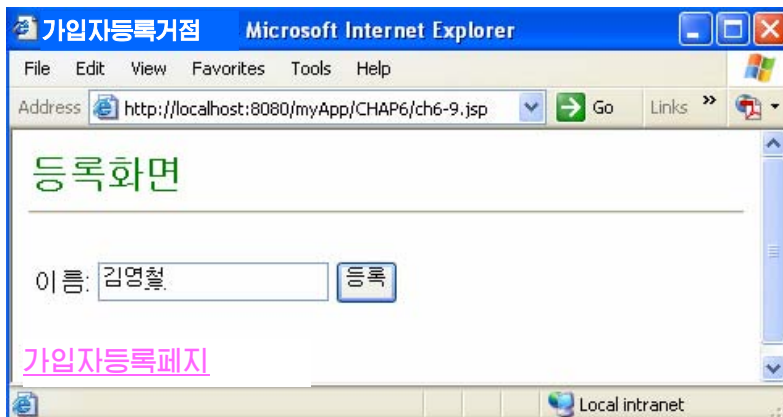



그림 6-7. 실례 6-9의 실행화면

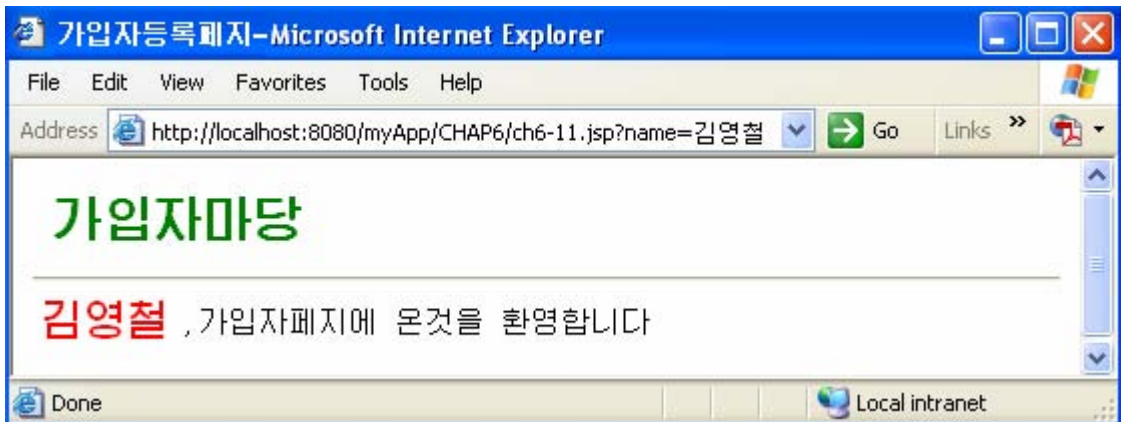



그림 6-8. 실례 6-11의 실행화면

제7장. 자료기지의 응용

제1절. 자료기지의 개요

어떤 기업소의 직원관리자료기지를 실례로 고찰하자. 그 자료구조를 그림 7-1에서 보여준다. 한개의 자료기지(Database)에는 자료표들이 있으며 자료표에는 마당들이 있다. 한개의 마당(Field)에는 고정된 자료형의 값을 보존한다. 또한 같은 자료표(Table)의 모든 마당은 고정된 마당첨수값(Index)들을 가지는데 제일 왼쪽마당의 첨수값은 1, 그 다음값은 2,…등으로 표시된다. 서로 같은 첨수값마당은 자료표에서 한개의 렬(Column)을 이룬다. 자료마당들이 한데 모여 한개의 레코드(Record)를 구성하는데 행(Row)이라고도 한다.(표 7-1)

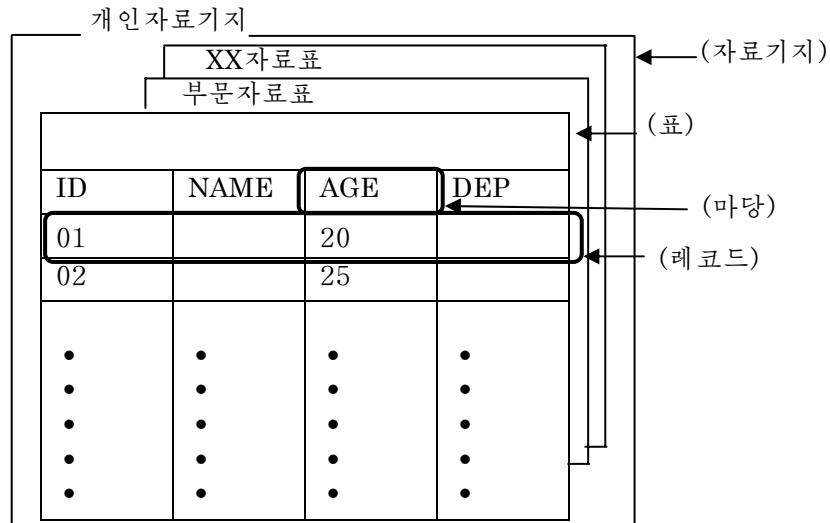



그림 7-1. 자료기지구조

표 7-1. 자료표형식

ID	NAME	AGE	DEP
02	김영철	25	회계부

또한 모든 레코드들이 모여 하나의 자료표(Table)를 구성한다. 보통 많은 자료를 보존하여야 할 때 우선 자료를 분석하여 분류한 다음 표(Table)의 형태로 이 자료들을 보존하여야 한다. 만일 자료를 분류하여 보존하지 않는다면 표의 구성을 복잡하게 만들뿐 아니라 검색효률도 저하시킬수 있다.

제2절. SQL언어

이 절에서는 SQL(Structure Query Language)언어에 대하여 설명한다. SQL언어는 구조화된 질문언어로서 주로 자료기지의 내용에 접근하는데 쓰인다. 사용자의 편리를 도모하여 간단한 조작방법을 제공하는 동시에 강력한 기능도 갖추고있다. SQL언어는 질문(Query)만으로 구성되여있으며 새로운 자료표(Table)를 만들고 자료기지의 레코드(Record)들을 수정하거나 삭제하는 기능 등을 수행한다. 말하자면 SQL언어는 자료기지에 대한 모든 조작을 할수 있다.

SQL언어는 이미 많은 관계형자료기지들(례하면 Oracle, DB2, Sybase 등)의 표준질문언어로 되고있다.

7.2.1. SQL언어의 분류

SQL언어는 IBM에 의해 처음으로 제기되였다. IBM이 SQL언어를 발전시킨 주요목적은 관계형자료기지를 다루기 위해서였다. 당시 IBM은 연구실에서 《SystemR》라는 실험계획을 세우고 진행하였으며 많은 실험을 통하여 이것의 정확성을 검증하였다. 1987년에 ANSI와 ISO가 공동으로 ANSI SQL-92의 SQL언어표준을 발표하였다.

SQL언어는 자료정의언어(DDL:Data Definition Language)와 자료유지언어(DML:Data Maintenance Language)로 나눈다. 자료정의언어의 create명령은 새로운 자료표(Table)를 만든다. 또한 자료유지언어는 자료표의 내용을 수정하는데 쓰이는데 례들어 insert명령은 자료표에 하나의 레코드(Record)를 삽입한다.

7.2.2. SQL의 기본명령

여기서는 7개의 자주 쓰는 SQL기본명령(alter, create, delete, drop, insert, select, update)들을 소개한다.

① ALTER명령

자료표의 마당(Field)을 새로 추가하거나 삭제한다. 그 형식은 다음과 같다.

・자료표에서 마당의 추가

ALTER TABLE[자료표이름]

ADD 마당이름 마당의 자료형선정;

・자료표에서 마당의 삭제

ALTER TABLE[자료표이름]

DROP 마당이름;

실례로 《전화번호책》자료표에서 《전화걸기》라는 마당을 새로 추가하고 《주소》마당은 삭제하려면

ALTER TABLE 전화번호책
ADD 전화걸기 VARCHAR(10)NOTNULL;
ALTER TABLE 전화번호책
DROP 주소;

② CREATE명령

새로운 자료표(Table)를 작성한다. 그 형식은 다음과 같다.

CREATE TABLE[자료표이름]
(
마당 1 마당의 자료형선정,
마당 2 마당의 자료형선정,
… … …
마당 n 마당의 자료형선정
);

실례로 전화번호책이라면

CREATE TABLE 전화번호책
(
이름 VARCHAR(10)NOTNULL,
전화 VARCHAR(10)NOTNULL,
주소 CHAR(20)
);

③ DELETE명령

자료표로부터 조건에 맞는 레코드(Record)를 삭제한다. 그 형식은 다음과 같다.

DELETE FROM[자료표이름]
WHERE[조건식];

여기서 WHERE[조건식]은 있을수도 있고 없을수도 있다. 없으면 자료표전체를 삭제한다.

실례로

DELETE FROM 전화번호책;

이것은 전화번호책자료표에서 모든 레코드들을 삭제해버린다.

실례로

DELETE FROM 전화번호책
WHERE 이름=“김영철”;

이것은 이름이 《김영철》인 레코드를 자료표에서 삭제한다. 표 7-2에서 자주 사용하는 조건연산자에 대하여 보여준다.

표 7-2. 조건연산자

=	갈기
>	크기
<	작기
>=	크거나 갈기
<=	작거나 갈기
!=	갈지 않기
LIKE	문자렬비교

④ DROP명령

자료기지로부터 한개의 자료표를 삭제한다. 그 형식은 다음과 같다.

DROP[자료표이름];

실례로

DROP 전화번호책;

⑤ INSERT명령

새로운 레코드(Record)를 자료표에 추가한다. 그 형식은 다음과 같다.

INSERT INTO[자료표이름](마당이름,…)

VALUES(마당값,…);

실례로

INSERT INTO 전화번호책(이름,전화,주소)

VALUES(“김영철”, “5551234”, “평양”);

⑥ SELECT명령

SELECT명령은 자료표안의 레코드를 선택하는데 리용한다. 그 형식은 다음과 같다.

SELECT 마당이름 FROM [자료표이름]

WHERE[조건식];

실례로

SELECT * FROM 전화번호책

WHERE 주소=“남포”;

이것은 자료표에서 《주소》가 《남포》인 모든 레코드들을 선택한다. 여기서 《*》는 모든 마당의 선정을 의미한다.

실례로

SELECT * FROM 전화번호책

WHERE 이름=“김영철” AND 주소=“남포”;

이것은 이름이 《김영철》이면서 주소가 《남포》인 레코드들을 모두 선택한다.

실례로

SELECT * FROM 전화번호책

WHERE 이름 LIKE “김%”;

은 전화번호책에서 성이 《김》인 레코드의 값들을 선택한다.

⑦ UPDATE명령

이미 자료표에 있는 레코드를 수정한다.

UPDATE[자료표이름]

SET 마당이름=마당값, 마당이름=마당값,…

WHERE[조건식];

아래의 실례는 《김영철》의 전화번호와 주소를 《3331234》와 《남포》로 고친다.

UPDATE 전화번호책

SET 전화= “3331234” , 주소= “남포”

WHERE 이름= “김영철”;

제3절. 웨브자료기지접근

7.3.1. 웨브를 통하여 자료기지에 어떻게 접근하는가

아래의 그림은 웨브자료기지에 접근하는 흐름도이다.(그림 7-2) 봉사기측은 웨브봉사기와 자료기지봉사기로 구성되여있으며 의뢰기측은 단지 Java를 지원하는 열람기만으로 구성되여있다. 봉사기측의 웨브봉사기는 JSP프로그람을 집행하며 JSP프로그람에서 JDBC를 통해 자료기지봉사기와 련결하고 자료기지의 자료를 얻는다. JDBC를 통해 자료기지에 SQL명령을 적용하여 레코드의 추가, 삭제, 수정 등의 조작을 진행한다. 이것은 JDBC가 제공하는 클라스와 메쏘드에 의하여 실현된다. 웨브봉사기는 또한 얻은 자료기지결과를 HTML형식으로 앞단의 열람기에 보내는 역할을 한다. 만일 JDBC가 없으면 JSP프로그람은 전혀 자료기지와 련결할수 없다.

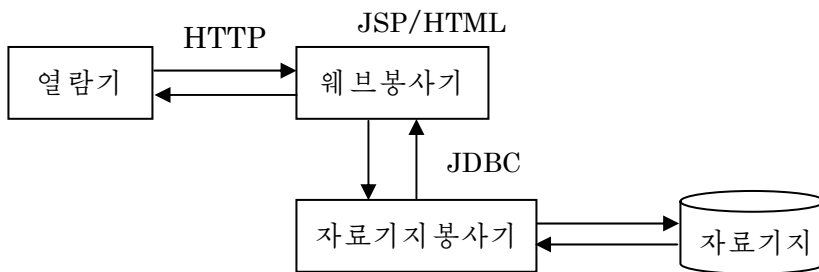



그림 7-2. 웨브자료기지흐름도

7.3.2. JDBC란

JDBC(Java DataBase Connectivity)는 Java언어가 관계형자료기지에 련결하고 조작하는 응용프로그람작성대면부(API)이다. JDBC는 클라스(Class)와 대면(Interface)으로 구성된다. 이 클라스와 대면들이 제공하는 메쏘드의 호출을 통하여 서로 다른 자료기지와 련결할수 있으며 자료기지에 대하여 SQL명령을 적용하여 그 결과를 얻을수 있다.

프로그람에서 JDBC API를 리용하자면 우선 체계에 JDK를 설치한 다음 프로그람머리부에 다음의 명령을 써야 한다. 즉

<%@page import= “java.sql.*” %>

Java에서는 JDBC의 모든 클라스와 대면들이 이 패키지에 속해있다. JSP에서 JDBC를 사용하기전에 java.sql패키지를 적재하지 않으면 JSP를 번역할 때 번역오유가 발생할수 있다.

7.3.3. JDBC구동프로그람의 분류

일반적으로 JDBC구동프로그람은 크게 4가지 류형으로 구분한다. 구동프로그람에 따라서 서로 사용방법이 다르며 자료기지에 련결하기전에 반드시 요구에 따라 적당한 구동프로그람을 선택하여야 한다.

1) JDBC-ODBC BRidge

다리형의 구동프로그람이다. 이 구동프로그람의 특징은 사용자측콤퓨터에서 먼저 ODBC구동프로그람을 설치한 다음 JDBC-ODBC의 호출방법을 리용하여 더 나아가서 ODBC를 통하여 자료기지에 접근하는것이다.

2) JDBC-Native API BRidge

다리형구동프로그람중의 하나이다. 류형 1과 같다. 이 구동프로그람 역시 우선 사용자콤퓨터에 구체적인 구동프로그람(ODBC와 류사하다.)을 설치한다. 그러면 다음에 JDBC-Native API다리의 전환에 의해 구동프로그람의 호출방법을 리용하여 자료기지에 접근하게 된다.

3) JDBC-middleware

이 구동프로그람의 우점은 사용자콤퓨터에 임의의 구동프로그람을 설치할 때 있게 되는 불리한 점을 제거한것이다. 봉사기측에 미들웨어(middleware)를 설치하기만 하면 미들웨어는 자료기지에 접근할 때 필요한 모든 전환을 책임진다.

4) Pure JDBC driver

이 구동프로그람은 제일 익숙된 JDBC구동프로그람이다. 사용자콤퓨터에 아무런 구동프로그람도 설치할 필요가 없을뿐아니라 봉사기측에 아무런 미들웨어를 설치할 필요가 없다. 자료기지에 접근하는 모든 조작은 직접 구동프로그람에 의하여 완성된다.

일반적으로 다리형구동프로그람을 사용하는것은 효과적인것이 못된다. 이러한 구동프로그람들은 순수한 Java언어로 개발한것이 아니므로 이식성에서 변화가 있을수 있으며 또

한 먼저 다른 구체적인 구동프로그람을 사용자콤퓨터에 설치할것을 요구하는 불편한 점을 가지고있다. 반대로 류형 3과 류형 4의 구동프로그람은 프로그람의 이식성을 높여줄뿐아니라 교차기반의 목적도 달성하게 한다. 이것은 다른 구동프로그람을 설치할 때의 불편을 감소시킨다. 그림 7-3은 JDBC구동프로그람의 접근구조를 보여주고있다.

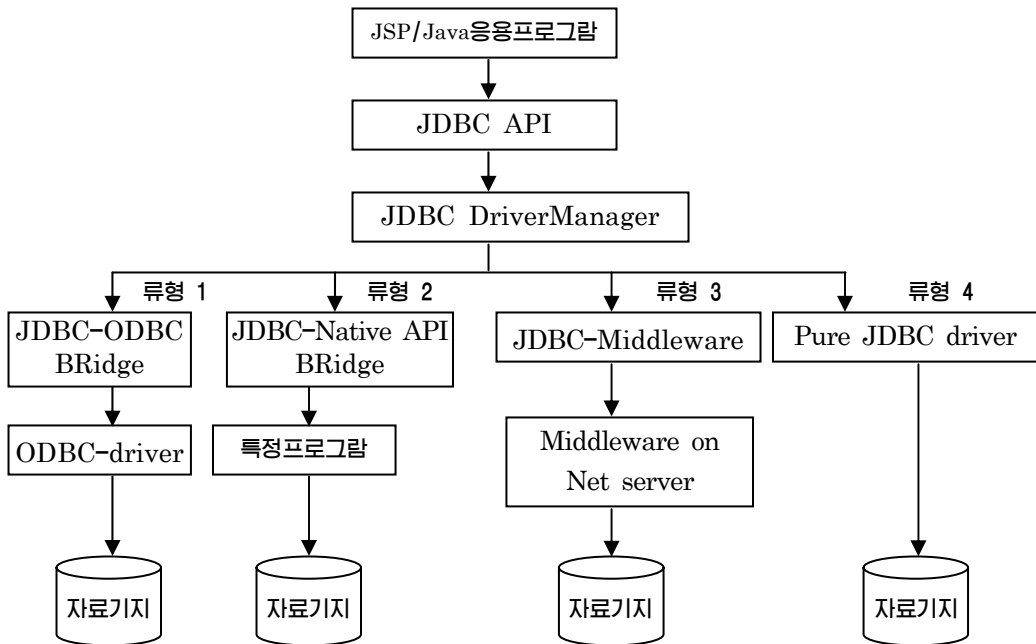



그림 7-3. JDBC구동프로그람들의 접근구조

7.3.4. JDBC의 클라스와 메쏘드

JDBC가 제공하는 클라스와 대면들은 많은데 여기에서는 자주 쓰는 클라스와 메쏘드들에 대하여 소개한다.

1) **DriverManager클라스**

이 클라스는 JDBC구동프로그람을 관리하는 클라스이다. JDBC구동프로그람을 사용하기 전에 우선 구동프로그람을 적재하고 DriverManager에 등록한 후에야 사용할수 있다. 프로그람에서는 Class.forName()메쏘드를 리용하여 실현할수 있다. DriverManager 클라스 역시 메쏘드를 제공하고있으며 자료기지와의 련결을 할수 있게 한다.(표 7-3)

표 7-3. DriverManager클라스의 메쏘드들

메 쏘 드	설 명
Static Connection getConnection (String url, String user, String password) throws SQLException	자료기지에 대한 접속을 진행 url:서식 jdbc:<subprotocol>:<subname> user: 자료기지에 련결하는 사용자이름 Password: 자료기지에 련결하는 통과암호
Static Driver getDriver(String url)throws SQLException	이미 DriverManager에 등록한 구동프로그람에서 URL이 지정한 자료기지를 열수 있는 구동프로그람을 찾는다.

아래의 실례는 자료기지와 어떻게 련결하는가를 보여준다.

실례 7-1

jsp(자료기지와의 련결)

```
<%@page language=“java”%>
<%@page import= “java.sql.*”%>
<%@page contentType= “text/html;charset=big5”%>
<HTML>
<HEAD>
<TITLE>레코드(Record)추가실례</TITLE>
</HEAD>
<BODY bgcolor=“#FFFFFF”>
<%
  String driver=“org.git.mm.mysql.Driver”;
  String url=“jdbc:mysql://localhost:3306/mydata”;
  String user=“root”;
  String password=“123456”;
  try
  {
    class.forName(driver);
  }
  catch(exception e)
  {
    out.println(“구동프로그람을 적재할수 없습니다:”+driver);
    e.printStackTrace();
  }
  try
  {
    Connection con=DriverManager.getConnection(url,user,password);
    Statement   smt=con.createStatement();
    smt.executeUpdate("INSERTinto phonebook(name,phone,addr,birth)"
```

```
      + "values( '김영철' ,  '4567890' ,  '평천구역' ,  "+"  '75.08.15' )" );
     con.close();
   }
   catch(SQLException SE)
   {
     SE.printStackTrace();
   }
%>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 MySQL자료기지와 어떻게 련결하는가를 보여주고있다.

우선 JDBC가 제공하는 클라스와 메쏘드를 사용해야 하므로 프로그람의 머리부에 아래의 행을 추가한다. 그렇지 않으면 번역할 때 오유를 발생할수 있다.

<%page import= "java.sql.*" %>

다음 행 《<%page content= "text.html;charset=Big5" %>》은 JSP홈페지에서 자료기지에 접근할 때 조선글을 리용하려는 경우 머리부에 추가하여야 한다.

또한 《Class.forName(driver);》은 java.lang.Class클라스가 제공하는 표준메쏘드 forName()을 통하여 DriverManager에 등록하고 사용하려는 JDBC구동프로그람을 적재한다. 여기서는 MySQL의 JDBC구동프로그람을 리용한다.

만일 IDBC-ODBC BRidge를 사용하려면 아래와 같은 행을 추가해야 한다. 즉

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

다음은 자료기지와의 련결을 하여야 한다. 즉

Connection con=DriverManager.getConnection(url,user,password);

여기서 URL은 자료기지의 위치를 지정하고있으며 user는 자료기지에 련결하는 사용자이름을 지정한다. 또한 password는 자료기지에 련결할 때의 통과암호를 지정한다. URL에 관한 지정방법은 일반적으로 아래와 같다. 즉

jdbc=<subprotocol>:<subname>

이 실례에서 URL를 리용하여 지정한 자료기지의 위치는 《jdbc:mysql://localhost:3306/ mydata" 》이다.

여기서 <subprotocol>부분은 mysql이 련결하려는 자료기지류형을 지정하며 <subname>에서는 mysql자료기지의 위치가 《localhost:3306》이다는것을 지정한다. 또한 사용하여야할 자료기지이름(mydata)을 지정할수도 있다.

만일 URL에서

jdbc:odbc:mydata

라고 하면 이때에는 jdbc:odbc다리를 통하여 mydata자료기지에 련결된다.

DriverManager.getConnection()메쏘드를 호출한 다음에는 자료기지와 련결하는 Connection클라스의 객체들을 리용하여야 한다.

프로그람의 실행결과를 그림 7-4에서 보여준다.

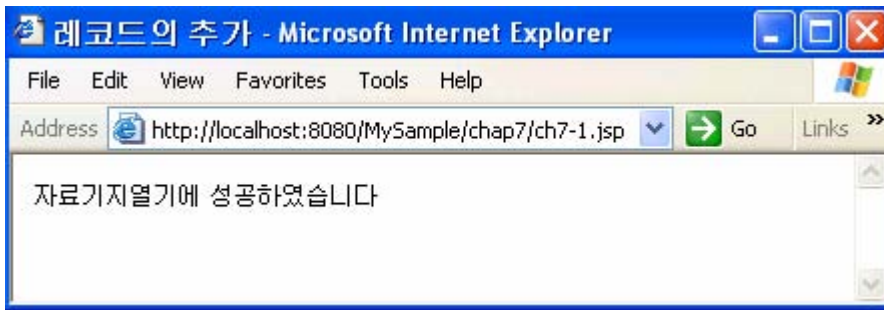



그림 7-4. 실례 7-1의 실행결과

2) Connection클라스

Connection클라스의 객체들은 JSP/Java자료기지프로그람과 자료기지사이의 련결을 유지한다. Connection클라스가 제공하는 메쏘드들을 통하여 3개의 클라스 Statement클라스, PreparedStatement클라스, DatabaseMetaData클라스를 만들수 있다. 표 7-4에서 이에 대한 구체적인 설명을 한다.

표 7-4. Connection클라스의 메쏘드들

메 쏘 드	설 명
Statement createStatement() throws SQLException	Statement클라스의 객체를 창조
Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException	Statement클라스의 객체를 창조
DatabaseMetaData getMetaData() throws SQLException	DatabaseMetaData클라스의 객체를 창조
PreparedStatement prepareStatement(String sql) throws SQLException	PreparedStatement클라스의 객체를 창조
boolean getAutoCommit() throws SQLException	Connection클라스의 객체의 자동끝내기 상태를 귀환
void setAutoCommit(boolean autoCommit) throws SQLException	Connection클라스의 객체의 자동끝내기 상태를 설정
void commit() throws SQLException	자료기지에 대하여 레코드의 추가, 삭제, 수정을 하는 조작을 확정

void rollback() throws SQLException	자료기지에 대하여 레코드의 추가, 삭제, 수정을 하는 조작을 취소
void close() throws SQLException	Connection클라스의 객체의 자료기지에 대한 련결을 결속
boolean isClosed() throws SQLException	Connection클라스의 객체의 자료기지에 대한 련결이 이미 결속되였는가를 판단

3) Statement클라스

Statement클라스가 제공하는 메쏘드를 통하여 표준적인 SQL명령을 리용할수 있으며 자료기지에 대하여 직접 레코드의 추가, 삭제, 수정 조작을 진행할수 있다.(표 7-5)

표 7-5. Statement클라스의 메쏘드들

메 쏘 드	설 명
ResultSet executeQuery(String sql) throws SQLException	SQL명령 SELECT를 사용하여 자료기지의 레코드조사작업을 진행
int executeUpdate(String sql) throws SQLException	SQL명령 INSERT, DELETE, UPDATE를 사용하여 자료기지에 대해 레코드의 추가, 삭제, 수정작업을 진행
void close() throws SQLException	Statement클라스의 객체의 자료기지에 대한 련결을 결속

아래의 프로그람은 mydata자료기지의 phonebook자료표에 하나의 레코드를 추가하는 실례이다.

실례 7-2

jsp(레코드의 추가)

```
<%@page language= “java” %>
<%@page import= “java.sql.*” %>
<%@page contentType= “text/html;charset=GB2312” %>
<HTML>
<HEAD>
<TITLE>레코드(Record)추가실례</TITLE>
</HEAD>
<BODY bgcolor= “#FFFFFF” >
<%
  String driver= “org.git.mm.mysql.Driver” ;
  String url= “jdbc:mysql://localhost:3306/mydata” ;
```

```
   String user= "root" ;
   String password= "123456" ;
   try
   {
      Class.forName(driver);
   }
   catch(Exception E)
   {
      out.println( "구동프로그람을 열수 없습니다:" +driver);
      E.printStackTrace();
   }
   try
   {
      Connection con=DriverManager.getConnection(url,user,password);
      Statement smt=con.createStatement();
      smt.executeUpdate( "INSERT into phonebook(name,phone,addr,birth)"
      + "values( '김영철' , '4567890' , '평천구역' ," + " '75.08.15' )" );
      smt.close();
      con.close();
   }
   catch(SQLException SE)
   {
      SE.printStackTrace();
   }
%>
</BODY>
</HTML>
```

프로그람설명

Connection클라스의 객체를 얻은 다음 Connection클라스가 제공하는 메쏘드 createStatement()를 리용하여 Statement클라스의 객체를 창조한다. 계속하여 아래의 코드를 보기로 하자.

```
smt.executeUpdate
( "INSERT into phonebook(name,phone,addr,birth)"
+ "values( '김영철' , '4567890' , '평천구역' ," + " '75.08.15' )" );
```

이것은 표준적인 SQL명령 INSERT의 문자렬파라메터를 executeUpdate()메쏘드에 보내고 Connection클라스의 객체에 의하여 유지되는 자료기지와 련결하여 SQL명령을 자료기지에 줌으로써 레코드의 추가조작을 진행할수 있도록 한다.(그림 7-5)

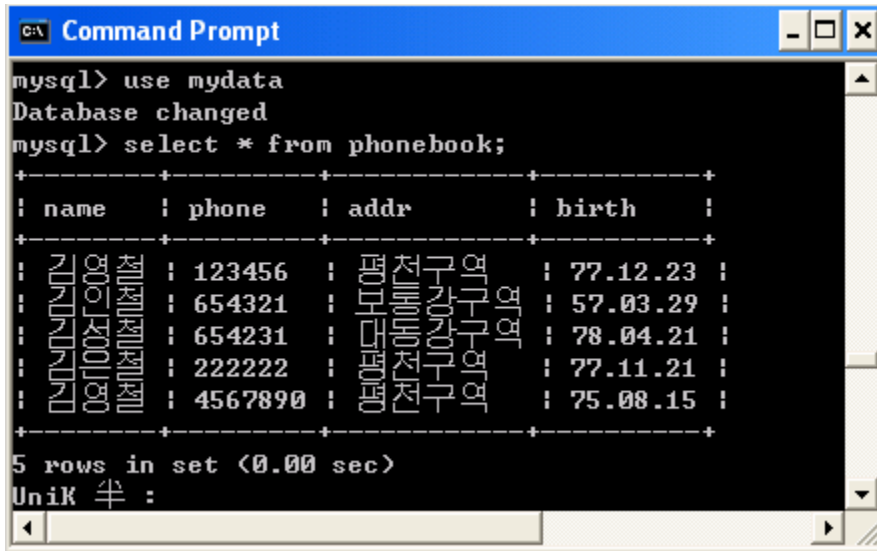



그림 7-5. 실례 7-2의 실행결과

executeQuery()메쏘드와 executeUpdate()메쏘드의 사용방식은 서로 류사하다. 이 메쏘드들의 파라메터로는 SQL명령을 리용한다.

기본적인 차이는 executeQuery()메쏘드가 SQL명령 SELECT를 사용하여 자료기지에 대한 검색을 진행할 때 리용된다는것이다. 또한 RecordSet클라스의 객체를 창조하여 검색결과를 보존할수도 있다. SELECT명령 이외의 다른 SQL명령들인 INSERT, DELETE, UPDATE, DROP 등 자료기지조작명령들도 리용한다.

4) PreparedStatement클라스

PreparedStatement클라스와 Statement클라스의 차이점은 PreparedStatement클라스의 객체에 반입되는 SQL명령을 우선 편집하기 좋게 준비하여 사용한다는데 있다. 그러므로 하나의 SQL명령을 여러번 리용할 때 PreparedStatement클라스를 쓰면 Statement클라스를 쓸 때보다 효률적이다.(표 7-6)

표 7-6. PreparedStatement클라스의 메쏘드들

메 쏘 드	설 명
ResultSet executeQuery() throws SQLException	SQL명령 SELECT를 사용하여 자료기지에 대해 레코드조사작업을 진행한다.
int executeUpdate() throws SQLException	SQL명령 INSERT, DELETE, UPDATE를 사용하여 자료기지에 대해 레코드의 추가, 삭제, 수정작업을 진행한다.
ResultSetMetaData getMetaData() throws SQLException	ResultSet클라스의 객체에서 마당과 관련한 정보를 얻는다.

void setInt(int parameterIndex, int x) throws SQLException	PreparedStatement클라스의 객체의 입력파라메터에 옹근수형수값을 설정한다.
void setFloat(int parameterIndex, float x) throws SQLException	PreparedStatement클라스의 객체의 입력파라메터에 류점수형수값을 설정한다.
void setNull(int parameterIndex,int sqlType) throws SQLException	PreparedStatement클라스의 객체의 입력파라메터에 NULL형수값을 설정한다.
void setString(int parameterIndex,String x) throws SQLException	PreparedStatement클라스의 객체의 입력파라메터에 문자렬형수값을 설정한다.
void setDate(int parameterIndex,date x) throws SQLException	PreparedStatement클라스의 객체의 입력파라메터에 날자형수값을 설정한다.
void setTime(int parameterIndex,Time x) throws SQLException	PreparedStatement클라스의 객체의 입력파라메터에 시간형수값을 설정한다.

아래의 프로그람은 PreparedStatement클라스의 객체를 사용하여 자료기지의 레코드를 갱신하는 실례이다.

실례 7-3

jsp(레코드의 수정(UPDATE))

```
<%@page language= “java” %>
<%@page import=  “java.sql.*” %>
<%@page contentType=  “text/html;charset=GB2312” %>
<HTML>
<HEAD>
<TITLE>레코드(Record)수정실례</TITLE>
</HEAD>
<BODY bgcolor= “#FFFFFF” >
<%
        String driver  = “org.git.mm.mysql.Driver” ;
        String url= “jdbc:mysql://localhost:3306/mydata” ;
        String user= “root” ;
        String password=” 123456” ;
        try
        {
                Class.forName(driver);
        }
        catch(Exception E)
        {
                out.println( “구동프로그람을 열수 없습니다:” +driver);
                E.printStackTrace();
        }
```

```
        try
        {
            Connection con=DriverManager.getConnection(url,user,password);
                PreparedStatement
                    psmt= con.prepareStatement( “UPDATE phonebook”
                    + “set phone = ?,addr = ?” + “where name = ?”);
            psmt.setString(1,  “4561234”);
            psmt.setString(2,  “평성시”);
            psmt.setString(3,  “Tome”);
            psmt.executeUpdate();
            Psmt.close();
            con.close();
        }
        catch(SQLException SE)
        {
            SE.printStackTrace();
        }
%>
</BODY>
</HTML>
```

프로그람설명

우선 Connection클라스가 제공하는 PreparedStatement()메쏘드를 사용하여 PreparedStatement클라스의 객체를 창조한다. PreparedStatement클라스의 입력파라메터는 SQL명령에서 ?로 대신한다. 이때 입력파라메터의 형은 자료기지서식의 마당형과 일치하여야 한다. 다음의 executeUpdate()메쏘드를 사용하여 SQL명령이 자료기지의 레코드수정을 진행하도록 한다.

MySQL자료기지를 통하여 조사한 결과를 그림 7-6에서 보여준다.

그림 7-6. 실례 7-3의 실행결과

5) DatabaseMetaData클라스

DatabaseMetadata클라스는 자료기지의 모든 특성을 보존하고있다. 또한 많은 메쏘드들을 제공하여 필요한 정보들을 얻는다. 메쏘드들에 대한 내용을 표 7-7에 주었다.

표 7-7. DatabaseMetadata클라스의 메쏘드들

메 쏘 드	설 명
String getDatabaseProductName() throws SQLException	자료기지이름 얻기
String getDatabaseProductVersion() throws SQLException	자료기지의 판본번호 얻기
String getDriverName() throws SQLException	JDBC구동프로그람의 이름얻기
String getDriverVersion() throws SQLException	JDBC구동프로그람의 판본번호 얻기
String getURL() throws SQLException	자료기지를 련결하는 JDBC URL얻기
String getUserName() throws SQLException	자료기지에 등록한 사용자이름 얻기

실례 7-4

jsp(자료기지정보의 얻기)

```
<%@page language= “java” %>
<%@page import= “java.sql.*” %>
<%@page contentType= “text/html;charset=GB2312” %>
<HTML>
<HEAD>
<TITLE>자료기지정보얻기실례</TITLE>
</HEAD>
<BODY bgcolor= “#FFFFFF” >
<%
        String driver= “org.git.mm.mysql.Driver” ;
        String url= “jdbc:mysql://localhost:3306/mydata” ;
        String user= “root” ;
        String password= “123456” ;
        try
        {
                Class.forName(driver);
        }
        catch(Exception E)
        {
                out.println( “구동프로그람을 열수 없습니다:” +driver);
                E.printStackTrace();
        }
```

```
        try
        {
                Connection con=DriverManager.getConnection(url,user,password);
                DatabaseMetaData dbmd = con.getMetaData();
                out.println("자료기지이름:");
                out.println(dbmd.getDatabasePuoductName()+"<p>");
                out.println("자료기지의 판본번호:");
                out.println(dbmd.getDatabasePuoductVersion()+"<p>");
                out.println("JDBC구동프로그람의 이름:");
                out.println(dbmd.getDriverName()+"<p>");
                out.println("JDBC구동프로그람의 판본번호:");
                out.println(dbmd.getDriverVersion() + "<p>");
                out.println("JDBC URL:");
                out.println(dbmd.getURL()+"<p>");
                out.println("사용자등록번호:");
                out.println(dbmd.getUserName()+"<p>");
                con.close();
        }
        catch(SQLException SE)
        {
                SE.printStackTrace();
        }
%>
</BODY>
</HTML>
```

프로그람의 실행결과는 그림 7-7과 같다.

그림 7-7. 실례 7-4의 실행결과

6) ResultSet클라스

SELECT명령을 사용하여 자료기지에 대한 조사를 진행하면 조사결과가 얻어진다. 이 결과를 보관하는 기능을 수행하는것이 ResultSet클라스의 객체이다. ResultSet클라스는 실제적으로 일련의 메쏘드들을 가지고 표준적인 SQL명령을 사용하지 않고도 자료기지에 대하여 레코드의 추가, 삭제, 수정 조작을 할수 있다.

또한 ResultSet클라스의 객체는 레코드유표를 조작하는 역할을 한다. 레코드유표는 자료표(Table)에서의 어떤 레코드를 가리킨다. ResultSet클라스의 객체를 리용하면 레코드유표를 임의의 위치로 옮겨 마음대로 자료기지에 접근함으로써 프로그람의 효률을 높일수 있다.(표 7-8)

표 7-8. ResultSet클라스의 메쏘드들

메 쏘 드	설 명
boolean absolute(int row) throws SQLException	레코드유표를 지정한 레코드로 이동시킨다.
void beforeFirst() throws SQLException	레코드유표를 첫번째 레코드앞에로 이동시킨다.
void afterLast() throws SQLException	레코드유표를 마지막 레코드뒤에로 이동시킨다.
boolean first() throws SQLException	레코드유표를 첫번째 레코드로 이동시킨다.
boolean last() throws SQLException	레코드유표를 마지막 레코드로 이동시킨다.
boolean next() throws SQLException	레코드유표를 다음 레코드로 이동시킨다.
boolean previous() throws SQLException	레코드유표를 이전 레코드로 이동시킨다.
void deleteRow() throws SQLException	레코드유표가 가리키는 레코드를 삭제한다.
void moveToInsertRow() throws SQLException	레코드유표를 추가하려는 레코드에로 이동시킨다.
void moveToCurrentRow() throws SQLException	레코드유표를 기억된(remembered) 레코드에로 이동시킨다.
void insertRow() throws SQLException	하나의 레코드를 자료기지에 추가한다.
void updateRow() throws SQLException	자료기지의 한개의 레코드를 수정한다.
void update류형(int columnIndex,류형 x) throws SQLException	지정된 마당의 값을 수정한다.
int get류형(int columnIndex) throws SQLException	지정된 마당의 값을 얻는다.
ResultSetMetaData getMetaData() throws SQLException	ResultSetMetaData클라스의 객체를 얻는다.

아래의 프로그람은 ResultSet클라스의 객체가 제공하는 메쏘드를 사용하여 레코드유표를 이동시키고 레코드유표가 가리키는 레코드의 마당값을 얻어내는 실례이다.

실례 7-5

Jsp(레코드유표(Cursor))

```
<%@page language= "java" %>
<%@page import= "java.sql.*" %>
<%@page contentType= "text/html;charset=GB2312" %>
<HTML>
<HEAD>
<TITLE>레코드유표(Cursor)실례</TITLE>
</HEAD>
<BODY bgcolor= "#FFFFFF" >
<%
  String driver= "org.git.mm.mysql.Driver" ;
  String url= "jdbc:mysql://localhost:3306/mydata" ;
  String user= "root" ;
  String password= "123456" ;
  try
  {
    Class.forName(driver);
  }
  catch(Exception E)
  {
    out.println( "구동프로그람을 열수 없습니다:" +driver);
    E.printStackTrace();
  }
  try
  {
    Connection con=
    DriverManager.getConnection(url,user,password);
    Statement smt = con.createStatement();
    ResultSet rst = smt.executeQuery( "SELECT*from phonebook" );
%>
    <table width=" 50%" border=" 2" >
<%
    rst.beforeFirst(); //레코드유표를 첫째 레코드의 앞까지 이동
    out.println( "모든 레코드:" );
    while(rst.next()) //레코드유표를 다음 레코드까지 이동
    {
%>
    <tr>
       <td><%=rst.getString(1)%></td>
       <td><%=rst.getString(2)%></td>
       <td><%=rst.getString(3)%></td>
       <td><%=rst.getString(4)%></td>
    </tr>
```

```
<%
    }
%>
    </table><p>
<%
    rst.first(); //레코드유표를 첫째 레코드까지 이동
    out.println( "첫번째 레코드:" );
%>
    <table width = "50%" border= "2" >
      <tr>
        <td><%=rst.getString(1)%></td>
        <td><%=rst.getString(2)%></td>
        <td><%=rst.getString(3)%></td>
        <td><%=rst.getString(4)%></td>
      </tr>
    </table><p>
<%
    rst.last(); //레코드유표를 마지막 레코드까지 이동
    out.println( "마지막 레코드:" );
%>
    <table width= "50%"  border= "2" >
      <tr>
        <td><%=rst.getString(1)%></td>
        <td><%=rst.getString(2)%></td>
        <td><%=rst.getString(3)%></td>
        <td><%=rst.getString(4)%></td>
      </tr>
    </table><p>
<%
    rst.absolute(3);//레코드유표를 3번째 레코드까지 이동
    out.println( "3번째 레코드:" );
%>
    <table width= "50%" border= "2" >
      <tr>
        <td><%=rst.getString(1)%></td>
        <td><%=rst.getString(2)%></td>
        <td><%=rst.getString(3)%></td>
        <td><%=rst.getString(4)%></td>
      </tr>
    </table><p>
<%
    smt.close();
    con.close();
  }
  catch(SQLException SE)
```

```
    {
    SE.printStackTrace();
    }
%>
</BODY>
</HTML>
```

프로그람설명

Statement클라스의 메쏘드 executeQuery()를 사용하여 자료기지를 조사하며 결과를 ResultSet클라스의 객체에 보존한다.

ResultSet rst=smt.executeQuery(“SELECT * from phonebook”);

executeQuery()메쏘드는 phonebook자료표를 조사하여 결과를 ResultSet클라스의 객체에 보존한다. 여기서는 레코드유표를 이동시키는 beforeFirst(), first(), last(), absolute(), next()메쏘드를 리용한다. getstring()메쏘드를 사용하여 phonebook자료표에서 레코드유표가 가리키는 레코드의 4개 마당값을 얻는다.

명령문 while(rst.next())은 While순환과 next()메쏘드의 조합으로서 next()메쏘드를 호출하여 레코드유표를 다음 레코드에로 이동시킬 때 성공이면 true가 귀환된다. 이 방법에 의해 자료표의 매 레코드를 처리할수 있다. 마지막 레코드까지 처리한 다음에는 false가 귀환된다.

실행결과를 그림 7-8에서 보여준다.

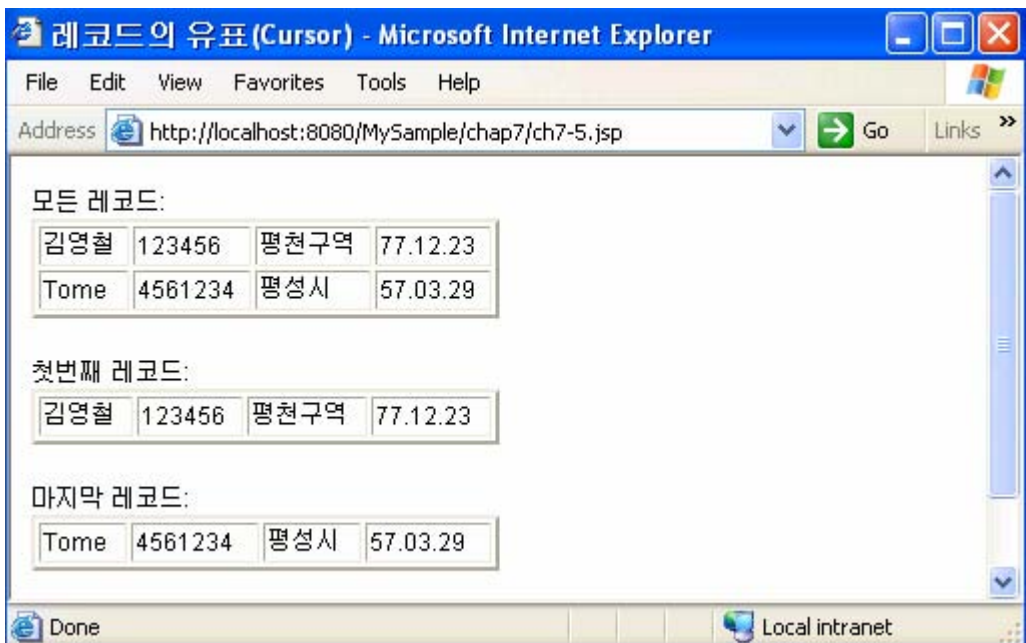



그림 7-8. 실례 7-5의 실행결과

7) ResultSetMetaData클라스

ResultSetMetaData클라스는 ResultSet클라스의 객체에서 표마당(Field)에 관한 모든 정보들을 보존하고있다.(표 7-9)

표 7-9. ResultSetMetaData클라스의 메쏘드들

메 쏘 드	설 명
int getColumnCount() throws SQLException	ResultSet클라스의 객체에서 표마당의 개수를 얻는다.
int getColumnDisplaySize(int column) throws SQLException	ResultSet클라스의 객체에서 표마당의 길이를 얻는다.
String getColumnName(int column) throws SQLException	ResultSet클라스의 객체에서 표마당의 이름을 얻는다.
String getColumnTypeName(int column) throws SQLException	ResultSet클라스의 객체에서 표마당의 형이름을 얻는다.
String getTableName(int column) throws SQLException	ResultSet클라스의 객체에서 이 표마당이 속한 자료표의 이름을 얻는다.
boolean isCaseSenstive(int column) throws SQLException	ResultSet클라스의 객체에서 표마당이 대소문자를 구분하는가를 검사한다.
boolean isReadOnly(int column) throws SQLException	ResultSet클라스의 객체에서 표마당이 읽기전용인가를 검사한다.

실례 7-6

jsp(마당정보의 얻기)

```
<%@page language= “java” %>
<%@page import=  “java.sql.*” %>
<%@page contentType=  “text/html;charset=GB2312” %>
<HTML>
<HEAD>
<TITLE>마당정보의 얻기</TITLE>
</HEAD>
<BODY bgcolor= “#FFFFFF” >
<%
   String driver = “org.git.mm.mysql.Driver” ;
   String url= “jdbc:mysql://localhost:3306/mydata” ;
   String user= “root” ;
   String password=” 123456” ;
   try
   {
```

```
    Class.forName(driver);
  }
  catch(Exception E)
  {
    out.println("구동프로그람을 열수 없습니다:"+driver);
    E.printStackTrace();
  }
  try
  {
    Connection con=DriverManager.getConnection(url,user,password);
    Statement  smt = con.createStatement();
    ResultSet  rst = smt.executeQuery("SELECT*from phonebook");
    ResultSetMetaData rsmd = rst.getMetaData();
    // getColumnCount()메쏘드를 사용하여 마당개수 얻기*
    out.println("자료표의 마당개수:");
    out.println(rsmd,getColumnCount()+"<p>");
    // getColumnName()메쏘드를 리용하여 마당이름얻기
    out.println("<table width='50%' border='2'>");
    out.println("<tr>");
    out.println("자료표의 마당이름:");
    For(int i=1;i<=rsmd.getColumnCount();i++)
    {
      out.println("<td>");
      out.println(rsmd.getColumnName(i));
      out.println("</td>");
    }
    out.println("</tr>");
    out.println("</table><p>");
    // getColumnTypeName()메쏘드를 리용하여 마당형이름얻기
    out.println("<table width=' '50%' border='2'>");
    out.println("<tr>");
    out.println("자료표의 마당류형이름:");
    for(int i=1;i<=rsmd.getColumnCount();i++)
    {
      out.println("<td>");
      out.println(rsmd.getColumnTypeName(i));
      out.println("</td>");
    }
    out.println("</tr>");
```

```
        out.println( “</table><p>” );
        // isReadOnly()메쏘드를 사용하여 마당이 읽기전용인가를 검사
        out.println( “table width= ‘50%’  border= ‘2’ >” );
        out.println( “<tr>” );
        out.println( “자료표의 마당은 읽기전용인가:” );
        for(int i=1;i<=rsmd.getCount();i++)
        {
            out.println( “</td>” );
            out.println(rsmd.isReadOnly(i));
            out.println( “</td>” );
        }
        out.println( “</tr>” );
        out.println( “</table>” );
        smt.close();
        con.close();
    }
    catch(SOLException SE)
    {
        SE.printStackTrace();
     }
%>
</BODY>
</HTML>
```

프로그람설명

프로그람에서 ResultSetMetaData클라스가 제공하는 메쏘드들을 사용하여 기본적으로 마당정보를 얻는다. 마당정보는 마당의 개수, 이름, 형 그리고 마당이 읽기전용인가를 포함하고있다. 여기서 꼭 알고 넘어가야 할 문제가 있다.

첫째는 for(int i=1;i<=rsmd.getColumnCount();i++)와 같이 for순환과 getColumnCount()메쏘드를 사용하여 어떤 레코드의 모든 마당에 접근할수 있다는것이며 다른 하나는 마당의 첨수값은 1부터 시작하여야 한다는것이다.

만일 for순환을 다음과 같이 설정한다면 for(int i=0;i<rsmd.getColumnCount() ;i++)는 집행과정에 오유를 발생시킬수 있다. 실행결과는 그림 7-9에서 보여준다.

그림 7-9. 실례 7-6의 실행결과

제8장. JSP의 응용실례

제1절. 다기능 망열람자계수기

이 장에서는 거점상에서 자주 보게 되는 홈페지의 기능들을 학습한다. 현재 일반 거점들에서 가장 많이 보게 되는것은 망열람자계수기이다. 이 절에서는 JSP를 리용하여 다기능의 망열람자계수기를 구성해보자.

여기서 계수기는 MySQL자료기지와 결합하여 망열람자가 이 거점에 대한 이번 달의 몇번째 망열람자로 되는가, 어느날 어느 시각에 방문하였는가를 그림으로 현시하여 망열람자의 동태를 알려주는 기능을 수행한다.

이 계수기는 몇개의 프로그람으로 구성된다.

- count.jsp: 거점의 첫번째 페지로서 사용자가 오늘 거점을 방문한 열람자중에서 몇번째 망열람자로 되는가를 현시한다.
- search.jsp: 사용자가 알아보려는 시간별망열람자수통계를 도표로 현시한다.
- opendata.jsp: 자료기지를 열고 접속하는 프로그람.

이 실례에서는 2개의 자료표 total과 counter를 리용하여 거점에서 망열람 총인원수와 매일 매시각의 망열람자수를 기록한다.

우선 자료표 total을 만든다.

```
mysql>create table total(
->total int(6)
->);
```

이 자료표 total은 오직 한개의 《total》마당만을 가지며 자료표를 만든 다음 MySQL명령 《insert into total set total=0;》을 실행하여 현재의 총망열람자수를 0으로 설정한다.

계속하여 또 다른 자료표 counter를 만든다.

```
mysql>create table counter(
->date char(6),
->today int(6),
->first int(6),
->second int(6),
->third int(6),
->forth int(6),
->fifth int(6),
->sixth int(6)
->);
```

이 자료표의 마당 《date》는 문자형자료이며 다른 마당들은 수값형자료들이다. 앞으

로 프로그람에서 《date》마당을 리용하여 날자를 기록한다. 《today》마당은 이날의 총망열람인원수를 기록한다. 그밖에 《first》, 《second》, 《third》, 《forth》, 《fifth》, 《sixth》마당은 각각 00~04, 04~08, 08~12, 12~16, 16~20, 20~24시간구간안의 망열람자수를 나타낸다.(표 8-1)

표 8-1. counter자료표

date	today	first	second	third	forth	fifth	sixth
m11d2	43	7	15	3	6	1	11
m11d3	60	15	16	3	18	4	4
m11d7	20	3	6	4	5	0	2
m11d10	557	98	241	114	43	11	50
m11d13	2	2	0	0	0	0	0

《m11d2》은 《date》마당의 자료로서 망열람한 날이 11월 2일이라는것을 의미하며 이날 망열람총인원수는 43명으로서 이것은 《today》마당에 기록된다.

다음은 우의 내용들을 종합하여 완성된 프로그람에 대하여 보기로 하자.

실례 8-1

counter.jsp

```
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@include file="opendata.jsp"%>
<%
    request.getSession(true);
    int month,day,hour,today,total,select;
    String time;
    String errmsg=request.getParameter("errmsg");
    GregorianCalendar calendar;
    calendar=new GregorianCalendar();
    month=calendar.get(Calendar.MONTH)+1;
    day=calendar.get(Calendar.DAY_OF_MONTH);
    hour=calendar.get(Calendar.HOUR_OF_DAY);
    time="m"+String.valueOf(month)+"d"+String.valueOf(day);
    sql="select * from counter where date='"+time+"'";
    rs=smt.executeQuery(sql);
    if(!rs.next())
    {
        sql="insert into counter(date,today,first,second,third,forth,fifth,sixth)values
```

```
            ('"+time+"',0,0,0,0,0,0,0)";
        rs=smt.executeQuery(sql);
    }
    sql="select * from counter where date='"+time+"'";
    rs=smt.executeQuery(sql);
    today=rs.getInt();
    select=(int)Math.floor(hour/4)+1;
    sql="select * from total";
    rs=smt.executeQuery(sql);
    total=rs.getInt(1);
    if(session.isNew())
    {
        today++;
        switch(select)
        {
               case 1:sql="update counter set today="+today+",first=first+1
                           where date='"+time"'";smt.executeQuery(sql);
                         break;
               case 2:sql="update counter set today="+today+",second=second+1
                           where date='"time+"'";smt.executeQuery(sql);
                         break;
               case 3:sql="update counter set today="+today+",third=third+1
                           where date='"+time"'";smt.executeQuery(sql);
                         break;
               case 4:sql="update counter set today="+today+",forth=forth+1
                           where date='"time+"'";smt.executeQuery(sql);
                         break;
               case 5:sql="update counter set today="+today+",fifth=fifth+1
                           where date='"+time"'";smt.executeQuery(sql);
                         break;
               case 6:sql="update counter set today="+today+",sixth=sixth+1
                           where date='"time+"'";smt.executeQuery(sql);

        }
    total++;
    sql="update total set total=total+1";
    smt.executeQuery(sql);
    smt.close();
    con.close();
%>
```

프로그람설명

이 프로그람은 앞부분이 자료의 접근과 수정조작이고 뒤부분이 망열람자수에 대한 정보를 현시하는 부분이다.

① 우선 Gregorian Calendar클라스를 사용하여 현재의 시각을 표시하는 월, 일, 시간을 얻어 각각 변수 month, day, hour에 넣는다.

《time= “m” +String.valueOf(month)+ “d” +String.valueOf(day);》는 현재 날자를 얻는 행이다. 실례로 오늘이 11월 2일이라면 《time=m11d2》로 된다. 이 변수를 리용하여 자료표에서 이날의 망열람자수를 찾는다.

② 자료기지의 date마당에서 time과 같은 날자를 찾고 그날의 망열람자수를 읽어들인다. 만일 이 자료를 찾을수 없으면 즉 사용자가 이날의 첫번째 망열람자이라면 아래의 프로그람코드를 리용하여 counter자료표에 이날의 새 자료로 기록한다. 즉

```
if(!rs.next())
{
sql= “insert into counter(date,today,first,second,third,forth,fifth,sixth)values
                        ( ‘ “+time” ’ ,0,0,0,0,0,0,0)”;
rs=smt.executeQuery(sql);
}
```

③ 명령문 《if(session.isNew())》은 사용자가 처음으로 홈페지에 들어갔는가를 판단하는 기능을 수행하는데 만일 조건이 성립하면 곧 망열람자수를 루계계산하는 코드를 집행한다. 조건이 성립하지 않으면 사용자가 refresh단추를 찰칵해도 망열람자수를 증가시킬수 없다.

④ 《select=(int)Math.floor(hour/4)+1;》은 사용자의 망열람시간토막을 계산한다. 《hour》는 현재 시간의 《시》를 표시하며 《Math.floor()》는 무조건 소수점자르기하여 옹근수를 얻는다. 실례로 현재 시간이 9시반이면 《select=(int)Math.floor(9/4)+1》의 결과는 3이다. 《select=3》은 사용자망열람시간토막이 3번째토막이다는것을 의미한다. 다음 switch판단명령문을 통하여 어느 시간토막에서 망열람수를 하나 증가시켜야 하는가를 결정한다.

실례 8-2

counter.jsp(계속)

```
<HTML>
<TITLE>인원수통계</TITLE>
<p align=center><img src=zct.jpg><BR>
<%
        String showcount=String.valueOf(total);
        out.print("당신은 이 거점의 ");
        for(int i=0;i<showcount.length();i++)
```

```
        out.print("<img src="+showcount.charAt(i)+".jpg>");
        out.print("번째 망열람자입니다<BR>");
        showcount=String.valueOf(today);
        out.print("당신은 오늘의 ");
        for(int i=0;i<showcount.length();i++)
        out.print("<img src="+showcount.charAt(i)+".jpg>");
        out.print("번째 망열람자입니다");
%>
<hr>
<center>
<form action=search.jsp method=Post>
        <table border=0>
        <tr>
            <td><font color=green>망열람인원수통계조사:</font></td>
                <td><select size=1 name=month>
                <option selected>1<option>2<option>3<option>4
                <option>5<option>6<option>7<option>8<option>9
                <option>10<option>11<option>12</select>월</td>
                <td><select size=1 name=day>
                <option selected>1<option>2<option>3<option>4
                <option>5<option>6<option>7<option>8<option>9
                <option>10<option>11<option>12<option>13<option>14
                <option>15<option>16<option>17<option>18<option>19
                <option>20<option>21<option>22<option>23<option>24
                <option>25<option>26<option>27<option>28<option>29
                <option>30<option>31</select>일</td>
            <td><input type=submit name=send value=조사></td>
        </tr>
        <tr>
                <td colspan=4 align=center>
                <a href=search.jsp?month=<%=month%>&day=<%=day%>>
                <font size=3>오늘의 매 시간구간 흐름통계보기</font></a></td>
        </tr>
        </table>
</form>
<font color=red size=4>
<%
   if(errmsg!=null)
   out.print(errmsg);
%>
</font>
</HTML>
```

이 프로그람에서는 for순환을 리용하여 length()와 charAt()메쏘드를 배합하고있으며 열람회수의 값에 대응하는 .jpg그림을 화면상에 현시하고있다.

또한 한개의 창문을 만들어 망열람자가 임의의 날의 흐름통계를 조사할수 있도록 하였다. 이 창문에 대한 처리프로그람은 search.jsp이다. 그림 8-1은 counter.jsp의 실행화면이다.

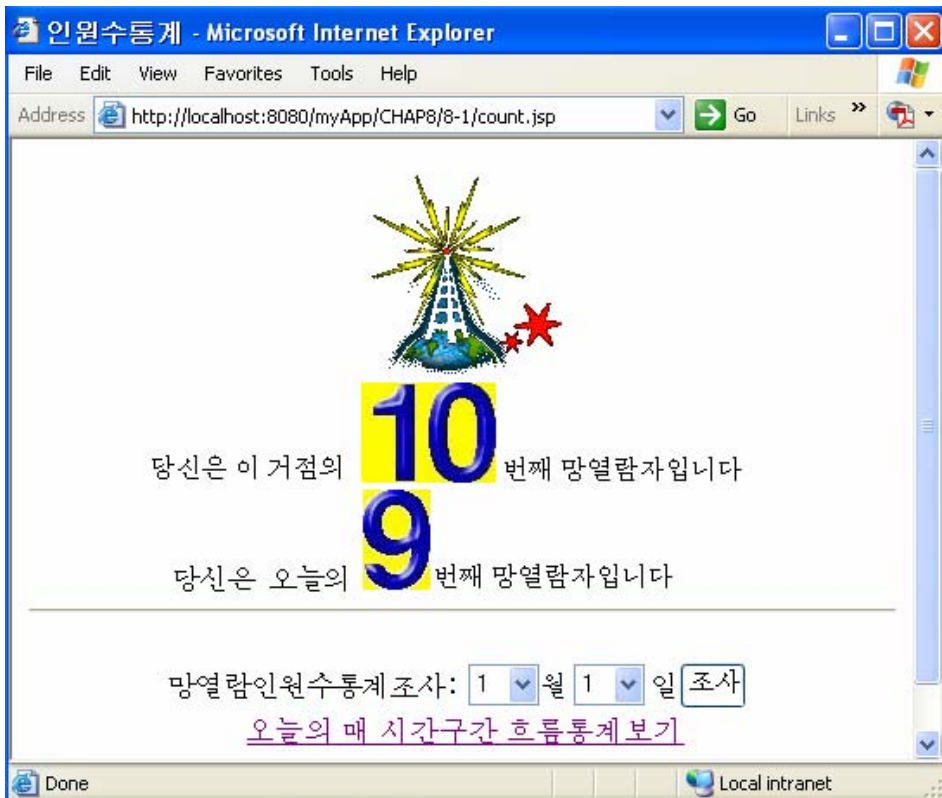



그림 8-1. counter.jsp의 실행결과

실례 8-3

opendata.jsp

```
<%
    Class.forName("org.gjt.mm.mysql.Driver");
    String url="jdbc:mysql://localhost:3306/mydata";
    Connection con=DriverManager.getConnection(url,"root","123456");
    Statement smt=con.createStatement();
    ResultSet rs;
    String sql;
%>
```

프로그람설명

앞에서 본 counter.jsp프로그람에서 《<%@include file= "opendata.jsp" %>》은 opendata.jsp를 열기 위한 코드이다. 먼저 JDBC를 사용하는 MySQL자료기지구동프로그람《org.git.mm.mysql.Driver》와 자료기지가 있는《url=jdbc:mysql://localhost:3306/mydata》를 정의하였다. 계속하여 자료기지에 접속하는 객체 《con》과 SQL명령을 집행하는 객체 《smt》를 생성한다. 자료기지에서 레코드모임을 얻는 객체 《rs》와 집행하여야 할 SQL명령의 문자렬변수 《sql》을 선언하였다.

이 프로그람에서는 사용자식별자는《root》로, 통과암호는《123456》로하여 MySQL자료기지에 접속한다.

실례 8-4

search.jsp

```
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%
int today,first,second,third,forth,fifth,sixth;
double p1,p2,p3,p4,p5,p6;
String month=request.getParameter("month");
String day=request.getParameter("day");
String time="m"+month+"d"+day;
sql="select * from counter where date='"+time+"'";
rs=smt.executeQuery(sql);
if(!rs.next())
{
        String errmsg="이날 망열람하지 않은 망열람자,다시 조사하십시오!";
        response.sendRedirect("count.jsp?errmsg="+errmsg);
}
else
{
        today=rs.getInt(2);
        first=rs.getInt(3);
        second=rs.getInt(4);
        third=rs.getInt(5);
        forth=rs.getInt(6);
        fifth=rs.getInt(7);
        sixth=rs.getInt(8);
        p1=(double)first/today*100;
        p2=(double)second/today*100;
        p3=(double)third/today*100;
        p4=(double)forth/today*100;
        p5=(double)fifth/today*100;
```

```
        p6=(double)sixth/today*100;
        smt.close();
        con.close();
%>
<HTML><TITLE>매 시간구간에서 망열람인원수통계</TITLE>
<BODY>
   <p align=center><font color=blue size=5><%=month%>월<%=day%>일 망열람기
   록통계표</font>
   <hr>
   <font color=green size=4>매 시간구간에서 망열람회수통계도</font>이날
   망열람총수:<%=today%><BR>
   00-04시 <img src=bar.jpg width=<%=first*10%>height=10><%=first%><BR>
   04-08시 <img src=bar.jpg width=<%=second*10%>height=10><%=second%><BR>
   08-12시 <img src=bar.jpg width=<%=third*10%>height=10><%=third%><BR>
   12-16시 <img src=bar.jpg width=<%=forth*10%>height=10><%=forth%><BR>
   16-20시 <img src=bar.jpg width=<%=fifth*10%>height=10><%=fifth%><BR>
   20-24시 <img src=bar.jpg width=<%=sixth*10%>height=10><%=sixth%><BR>
   <hr>
   <font color=green size=4>매 시간구간에서
   망열람비률통계도</font>총비률:100%<BR>
   00-04시 <img src=bar.jpg width=<%=p1*5%>height=10><%=p1%>%<BR>
   04-08시 <img src=bar.jpg width=<%=p2*5%>height=10><%=p2%>%<BR>
   08-12시 <img src=bar.jpg width=<%=p3*5%>height=10><%=p3%>%<BR>
   12-16시 <img src=bar.jpg width=<%=p4*5%>height=10><%=p4%>%<BR>
   16-20시 <img src=bar.jpg width=<%=p5*5%>height=10><%=p5%>%<BR>
   20-24시 <img src=bar.jpg width=<%=p6*5%>height=10><%=p6%>%<BR>
<%
   }
%>
   <hr>
   <center><a href=count.jsp>첫페지로 돌아가기</a>
</BODY>
</TITLE>
</HTML>
```

프로그람설명

이 프로그람에서는 막대도표를 리용하여 임의의 날의 매 시간구간의 망열람수와 비률을 현시하고있다. 이 방식은 망열람자와 망관리자가 보다 명백히 전체 거점의 흐름상황을 료해할수 있게 한다.

① 프로그람은 실행시작하자마자 변수 《time》을 리용하여 자료표로부터 선택한 날자에 해당되는 자료를 찾는다. 만일 이 자료가 없으면 오유정보가 출현한다.

② 변수 p1~p6은 매 시간구간안의 망열람자수에 대한 백분률을 보관하는데 쓰인다.

③ 매 시간구간의 막대도표를 현시하기 위하여 프로그람에서는 크기가 10*10pixels인 쪼각그림 《bar.jpg》를 사용하였다. 그 너비와 높이를 설정하여 요구하는 도형을 얻

을수 있다. 례하면 《<img src=bar.jpg width=100 height=10>》은 100*10pixels의 도형을 만든다. 우의 방법들을 사용하고 변수들을 배합하면 우리가 요구하는 통계도를 얻을수 있다.

08-12시 <img src=bar.jpg width=<%=p3*5%>height=10><%=p3%>%
12-16시 <img src=bar.jpg width=<%=p4*5%>height=10><%=p4%>%
16-20시 <img src=bar.jpg width=<%=p5*5%>height=10><%=p5%>%
20-24시 <img src=bar.jpg width=<%=p6*5%>height=10><%=p6%>%

우의 프로그람토막에서와 같이 어떤 날의 망열람자수를 배합하면 그림 8-2와 같은 결과를 얻을수 있다.

그림 8-2. search.jsp의 실행결과

아래에서는 이 응용실례의 결과를 보여주고있다.(그림 8-3)

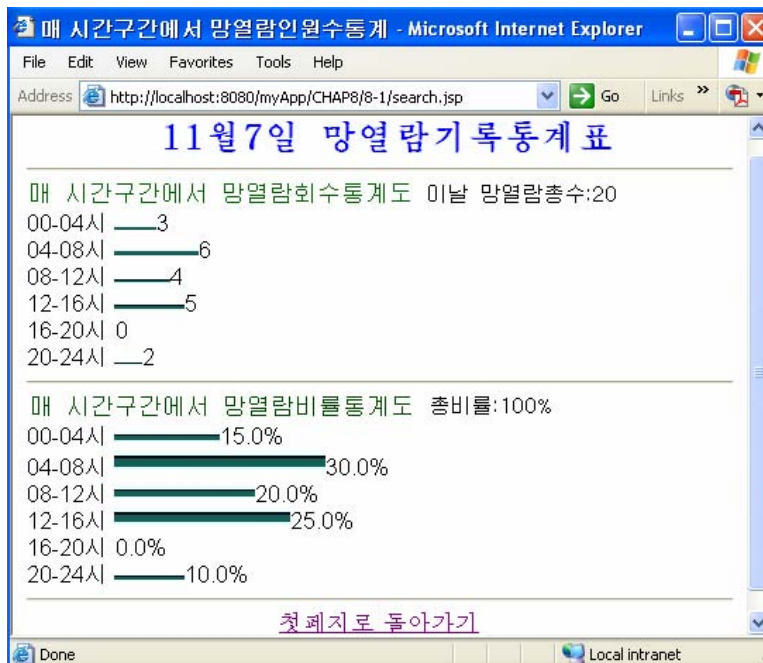



그림 8-3. 제1절의 실행결과

제2절. 게시판의 작성

게시판은 대규모의 거점으로부터 소규모의 개인홈페지에 이르기까지 반드시 갖추어야 할 구성부분중의 하나로 되고있다. 그것은 게시판을 리용하면 망열람자들이 편리하게 홈페지상에서 정보를 전달하고 게시할수 있기때문이다.

이 절에서는 게시판을 실제적으로 작성해 본다. 이 게시판은 게시기능을 제공하는외에 페지를 갈라 게시내용을 열람할수 있게 하려고 한다. 그리고 관리자가 자기의 전용화면에 들어가 게시물을 삭제할수 있는 《전문가의 게시판》이 되도록 한다.

게시판은 아래의 파일들로 구성된다.

- board.jsp: 게시판의 주프로그람이다. 사용자가 자기의 게시물을 볼수 있도록 하며 페지분할기능도 제공한다.
- message.htm: 사용자가 자료와 게시물을 입력하는 창문이다.
- opendata.jsp: MySQL자료기지에 접속하는데 쓰이는 프로그람이다.(실례 8-3참고)
- savememo.jsp: 사용자의 게시물을 자료기지에 보관한다.
- manager.jsp: 관리자전용의 게시판프로그람이다.
- password.jsp: 관리자의 등록화면이다.
- check.jsp: 관리자가 입력한 통과암호가 정확한가를 검증한다.
- delete.jsp: 게시물을 삭제하는 프로그람이다.
- convert.jsp: 조선글문자렬과 행바꾸기를 처리하는 자체정의함수파일이다.

프로그람작성시작 전에 우선 MySQL에서 message라는 자료표를 반드시 만들어야 한다. 그 내용은 아래와 같다.

```
mysql>create table message(
        ->name char(20),
        ->email char(40),
        ->subject char(60),
        ->time char(60),
        ->sex char(10),
        ->memo text,
        ->id int not null auto_increment,
        ->primary key(id)
        ->);
```

이 자료표에서 《name》, 《email》, 《subject》마당은 각각 사용자이름, 전자우편주소, 게시주제를 기록하는데 쓰이며 《time》마당은 게시시간을 기록하는 마당이다. 《sex》마당은 성별을 표현하는 쪼각그림파일이름을 보관한다. 《memo》는 게시내용을 기록하며 《id》는 등록번호이다. 이 응용실례의 대체적인 구조와 자료표의 내용을 료해한데 기초해서 이제는 프로그람의 내용을 보도록 하자.

실례 8-5

board.jsp

```
<HTML>
<TITLE>게시판</TITLE>
<BODY>
<%@page import="java.sql.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%
  int count=0,lastp,numf,numl,prep,nextp,pageno;
  if(request.getParameter("pageno")==null)
  pageno=0;
  else
  pageno=Integer.parseInt(request.getParameter("pageno"));
  sql="select * from message";
  rs=smt.executeQuery(sql);
  while(rs.next())
  count++;
  lastp=(int)Math.ceil((double)count/5);
  if(pageno==0||pageno>lastp)
  pageno=lastp;
  numf=pageno*5-4;
  numl=numf+4;
  if(pageno==1)
  prep=1;
  else
  prep=pageno-1;
  if(pageno==lastp)
  nextp=lastp;
  else
  nextp=pageno+1;
  sql="select * from message where id between "+numf+"and"+numl;
  rs=smt.executeQuery(sql);
%>
<font size=7 color=green>게시판</font>
<hr>
<center>
<form action=board.jsp method=Post>
    <table border=0>
        <tr>
            <td>현재페지번호:<font color=red><%=pageno%></font>/
```

```
            <font color=blue><%=lastp%></font></td>
            <td><a href=board.jsp?pageno=<%=prep%>>[이전페지]</a></td>
            <td><a href=board.jsp?pageno=<%=nextp%>>[다음페지]</a></td>
            <td><a href=board.jsp?pageno=1>[1페지]</a></td>
            <td><a href=board.jsp?[마지막페지]</a><td><td>페지번호입력:
            <input type=text size=3 name=pageno></td>
            <td><input type=text size=3 name=pageno></td>
            <td><input type=submit name=send value=보내기</td>
            <td><a href=password.jsp><font color=red size="5">
            <i>거점관리자전용</i></font></a></td>
        </tr>
    </table>
</form>
<%
  String name,email,subject,time,sex,memo;
  where(rs.next())
  {
        name=rs.getString(1);
        email=rs.getString(2);
        subject=rs.getString(3);
        time=rs.getString(4);
        sex=rs.getString(5);
        memo=rs.getString(6);
        out.print("<center>");
        out.print("<table border=1>");
        out.print("<tr><td bgcolor=yellow>이름</td><td>"+name+"</td></tr>");
        out.print("<tr><td bgcolor=yellow>E-mail</td><td>"+email+"</td></tr>");
        out.print("<tr><td bgcolor=yellow>시간</td><td>"+time+"</td></tr>");
        out.print("<tr><td bgcolor=yellow>주제</td><td>"+subject+"</td></tr>");
        out.print("<tr><td valign=top bgcolor=yellow>
                게시물</td><td>"+memo+"<img src="+sex+"></td></tr>");
        out.print("</table><p>");
  }
%>
<hr>
<center><a href=message.htm>게시물 요구</a>
<a href=board.jsp>게시물보기</a>
</BODY>
</HTML>
```

프로그람설명

《board.jsp》는 게시판의 주프로그람으로서 message자료표의 게시자료를 추출하여 한페지에 5개항목으로 된 자료를 게시한다. 아래에서 구체적으로 설명한다.

① message자료표의 자료를 추출하고 while순환명령문을 리용하여 총 몇개의 게시물이 올라있는가를 계산한 다음 그 값을 count변수에 넣는다. 그리고 마지막 페지의 페지번호는 《lastp》로 하며 《Math.ceil()》메쏘드를 사용할 때 마지막페지에서는 게시자료의 개수가 5을 넘을수 없으므로 페지수를 직접 얻을수 있다.

② 《pageno》는 페지번호를 의미하며 만일 사용자가 입력한 페지가 총 페지수보다 클 때 pageno를 lastp로 설정한다. 《prep》와 《nextp》는 각각 현재 페지의 전페지와 다음페지이다.

③ 《numf=pageno*5-4》는 임의의 페지에 있는 게시물의 첫번째 항목자료의 등록번호(message자료표의 id)이다. 즉 5번째 페지의 첫번째 항목자료등록번호는 21이다. 《numl》은 임의의 페지의 제일 마지막 항목자료의 등록번호이다.

④ 이 프로그람의 홈은 사용자가 직접 다음페지를 《<a href=board.jsp?pageno=>》의 형식으로 입력하여 pageno를 프로그람에 전송하게 하며 이 페지의 게시내용을 현시한다.

⑤ 프로그람은 사용자의 선택이나 입력에 따라 《pageno》를 얻을수 있으며 이 수를 리용하여 《numf》와 《numl》를 계산한다. 그리고 SQL명령을 집행하여 그사이의 게시자료들을 얻고 이것을 하나하나 열람기상에 현시한다.

실례 8-6

message.htm

```
<HTML>
<TITLE>게시물요구</TITLE>
<BODY>
  <font size=7 color=green>게시물요구</font>
  <hr>
  <center>
  <form action=savememo.jsp method=Get>
      <table border=1>
          <tr>
              <td bgcolor=yellow>이름:</td>
              <td><input type=text size=20 name=name></td>
          </tr>
          <tr>
              <td bgcolor=yellow>성별:</td>
              <td>남<input type=radio name=sex value=boy.gif checked>
```

```
                녀<input type=radio name=sex value=girl.gif></td>
            </tr>
            <tr>
                <td bgcolor=yellow>Email:</td>
                <td><input type=text size=40 name=email></td>
            </tr>
            <tr>
                <td bgcolor=yellow>주제:</td>
                <td><input type=text size=60 name=subject></td>
            </tr>
            <tr>
                <td valign=top bgcolor=yellow>게시물:</td>
                <td><textarea name=memo rows=4 cols=60></textarea></td>
            </tr>
            <tr align=center><td colspan=2>
                <input type=submit name=send value=제시물요구>
                <input type=reset value=다시쓰기></td>
            </tr>
        </table>
    </form>
    <hr>
    <center><a href=board.jsp>게시물보기란돌아가기</a>
</BODY>
</HTML>
```

프로그람설명

이 HTML파일은 그림 8-4와 같은 표를 만들어 사용자가 자료와 게시물을 입력할수 있도록 한다.

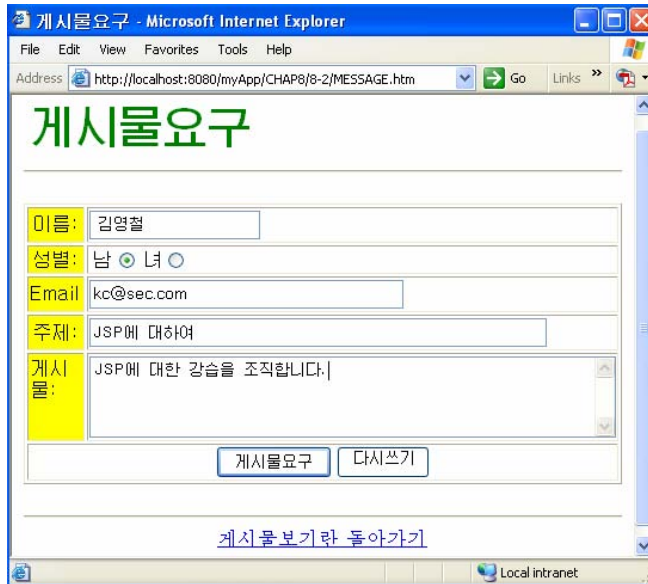



그림 8-4. Message.htm의 현시화면

이 표에서는 성별을 선택할수 있는데 남성을 선택하면 자료표의 《sex》는 《boy.gif》로 기입되며 녀성이면 《girl.gif》로 기입된다. 이것을 사용하면 게시물을 현시할 때 서로 대응하는 성별그림파일을 현시할수 있다.

실례 8-7

convert.jsp

```
<%!
   String convert(String str)
   {
        byte newstr[]=new byte[str.length()];
        for(int i=0;i<str.length();i++)
          newstr[i]=(byte)str.charAt(i);
          return new String(newstr);
   }
   String Replace(String str)
   {
        int index=0;
        while((index=str.indexOf("\n")) != -1)
                str=str.substring(0,index)+"<BR>"+str.substring(index+1);
   }
%>
```

프로그람설명

이 프로그람에서는 2개의 함수 convert와 Replace를 정의하였는데 이것들은 각각 조선글문자렬을 byte자료형으로 전환하고 사용자게시물에서의 행바꾸기를 하는데 쓰인다.

① convert함수는 사용자가 입력한 조선글을 byte자료형의 문자렬로 전환하는데 리용한다. MySQL내부자료기지는 byte자료형을 처리하고 MySQL의 JDBC구동프로그람도 자료를 주고받을 때 문자렬형(String)으로 전송하므로 사용자가 조선글을 입력하였을 때 Big5코드를 byte형으로 전환한 문자렬을 MySQL자료기지에 전송하여야만 정확한 현시와 처리를 할수 있다.

② Replace함수는 사용자가 입력한 게시물의 행바꾸기부호 《\n》(사용자가 [Enter]건을 누를 때)을 HTML의 행바꾸기꼬리표 《
》로 전환한다. 그리하여 게시물을 현시할 때 정확한 행바꾸기를 진행한다.

실례 8-8

savememo.jsp

```
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
   String name,email,subject,memo,sex;
   name=request.getParameter("name");
   sex=request.getParameter("sex");
   email=request.getParameter("email");
   subject=request.getParameter("subject");
   memo=request.getParameter("memo");
   if(name.length()==0||email.length()==0
         ||subject.length()==0||memo.length()==0)
   {
         out.print("<center><font color=red size=6>
                      입력마당은 빌수 없습니다</font>");
         out.print("<hr><a href=board.jsp>게시물보기란돌아가기</a>");
         out.print("<a href=message.htm>게시물요구</a>");
   }
   else
   {
         int count=0,year,month,day,hour,minute,second,lastp;
         String time;
         GregorianCalendar calendar;
```

```
        calendar=new GregorianCalendar();
        year=calendar.get(Calendar.YEAR);
        month=calendar.get(Calendar.MONTH)+1;
        day=calendar.get(Calendar.DAY_OF_MONTH);
        hour=calendar.get(Calendar.HOUR_OF_DAY);
        minute=calendar.get(Calendar.MINUTE);
        second=calendar.get(Calendar.SECOND);
        time=year+"년"+month+"월"+day+"일"+hour+":"+minute+":"+second;
        name=convert(name);
        time=convert(time);
        subject=convert(subject);
        memo=Replace(memo);
        memo=convert(memo);
        sql="insert into message set
            name='"+name+"',email='"+email+"',subject='"+
            subject"',time='"+time"',sex='"+sex+"',memo='"+memo+"'";
        smt.executeQuery(sql);
        while(rs.next())
                count++;
        lastp=(int)Math.ceil((double)count/5);
        response.sendRedirect("board.jsp?pageno="+lastp);
    }
%>
<HTML>
<TITLE>오유정보</TITLE>
</HTML>
```

프로그람설명

이 프로그람은 사용자가 《message.htm》홈에 입력한 자료를 자료기지에 넣는 실례이다.

① 《<%@include file= "convert.jsp" %>》은 convert.jsp파일을 적재한다.

② 《request.getParameter()》메쏘드를 사용하여 사용자가 입력한 매개 마당의 내용을 얻는다. 그리고 마당이 비였는가를 조사하여 만일 마당에 자료가 없다면 오유정보를 출력한다.

③ 현재 시간을 얻는다. 게시물현시시간을 표현하는 시간문자렬들을 조합하여 time변수에 넣는다.

④ convert함수를 사용하여 사용자가 조선글을 입력할수 있는 상태로 만든다. memo자료는 반드시 Replace함수를 사용하여 행바꾸기를 진행한다. 그리고 다시 MySQL명령을 집행하여 매 항목자료를 자료표의 서로 대응하는 마당에 넣는다.

⑤ 게시판의 마지막 페지번호 《lastp》를 계산한다. 다시말하여 사용자가 새로 추가한 게시물의 다음페지를 계산한다. 이 다음페지의 pageno를 borad.jsp에 보내여 게시물의 내용을 현시한다.

실례 8-9

password.jsp

```
<HTML>
<TITLE>망관리자의 등록화면</TITLE>
<BODY>
<font size=5 color=green>망관리자등록</font>
<font color=red>
<%
   String errmsg=request.getParameter("errmsg");
   if(errmsg!=null)
   out.print(errmsg);
%>
</font>
<hr>
<form action=check.jsp method=Post>
    <table border=0>
      <tr>
         <td>통과암호를 입력하십시오:</td>
         <td><input type=text size=6 name=password></td>
         <td><input type=submit name=send value=입력></td>
      </tr>
    </table>
</form>
<a href=board.jsp>게시물보기란돌아가기</a>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 사용자가 《거점관리자전용》을 선택하여 접속할 때 사용자통과암호를 입력할것을 요구한다. 만일 입력한 통과암호가 아래의 프로그람 check.jsp에서 맞지 않기때문에 오유로 판단하였을 때에는 자기 페지로 되돌아와 오유정보를 현시한다.

실례 8-10

check.jsp

```
<%
   String password=request.getParameter("password");
   if(password.equals("123456"))
      response.sendRedirect("manager.jsp");
   else
   {
      String errmsg="입력한 통과암호가 정확하지 않습니다!";
      response.sendRedirect("password.jsp?errmsg="+errmsg);
   }
%>
```

프로그람설명

이 프로그람은 사용자가 정확한 통과암호를 입력하였는가를 판단하는 실례이다. 통과암호가 《123456》으로 정확히 입력되였다면 관리자전용홈페지 manager.jsp에 들어가고 그렇지 않으면 오유통보문 《errmsg》를 password.jsp에 보낸다.

실례 8-11

manager.jsp

```
<html>
<title>게시판</title>
<body>
<%@page import="java.sql.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%
   int count=0,lastp,numf,numl,prep,nextp,pageno;
   if(request.getParameter("pageno")==null)
   pageno=0;
   else
   pageno=Integer.parseInt(request.getParameter("pageno"));
   sql=="select * from message";
   rs=smt.executeQuery(sql);
   while(rs.next())
   count++;
   lastp=(int)Math.ceil((double)count/5);
   if(pageno==0||pageno>lastp)
```

```
    pageno=lastp;
    numf=pageno*5-4;
    numl=numf+4;
    if(pageno==1)
    prep=1;
    else
    prep=pageno-1;
    if(pageno==lastp)
    nextp=lastp;
    else
    nextp=pageno+1;
    sql="select * from message where id between"+numf+"and"+numl;
    rs=smt.executeQuery(sql);
%>
<font size=7 color=green>게시판</font>
<font color=#008080>망관리자사용화면</font>
<hr>
<center>
<form action=board.jsp method=POST>
      <table border=0>
          <tr>
              <td>현재페지번호: <font color=red><%=pageno%></font>
              <font color=blue><%=lastp%></font></td>
              <td><a href=manager.jsp?pageno=<%=prep%>>[이전페지]<a/></td>
              <td><a href=manager.jsp?pageno=<%=prep%>>[다음페지]<a/></td>
              <td><a href=manager.jsp?pageno=1>[1페지]</a></td>
              <td><a href=manager.jsp>[마지막페지]</a></td><td>페지번호입력:
              <input type=text size=3 name=pageno></td>
              <td><input type=submit name=SEND value=보내기></td>
              <td><a href=password.jsp><font color=red size="5">
              <i>망관리자전용</i></font></a></td>
          </tr>
      </table>
</form>
<form action=delete.jsp?pageno=<%=pageno%>method=POST>
<%
  String  name, email, subject, time, sex, memo;
  int  id;
  while(rs.next())
  {
          name=rs.getString(1);
          email=rs.getString(2);
```

```
        subject=rs.getString(3);
        time=rs.getString(4);
        sex=rs.getString(5);
        memo=rs.getString(6);
        id=rs.getInt(7);
        out.print("<p align=left><input type=checkbox name=D"+id+
                  +"value=del><font color=red>"+id+"</font>번째 게시물 삭제");
        out.print("<center>");
        out.print("<table border=1>");
        out.print("<tr><td bgcolor=yellow>이름</td><td>"+name+"</td></tr>");
        out.print("<tr><td bgcolor=yellow>E-mail</td><td>"+
                  +email+"</td></tr>");
        out.print("<tr><td bgcolor=yellow>시간</td><td>"+
                  +time+"</td></tr>")
        out.print("<tr><td bgcolor=yellow>주제</td><td>"+
                  +subject+"</td></tr>");
        out.print("<tr><td valign=top bgcolor=yellow>게시물</td><td>"+
                  +memo+"<img src="+sex+"></td></tr>");
        out.print("</table><p>");
  }
%>
<input type=submit name=send value=삭제확정>
<input type=reset value=다시선택>
</form>
<hr>
<center><a href=message.htm>게시물요구</a>
<a href=manager.jsp>게시물보기</a>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 구조상 board.jsp와 거의 같다. 매 게시물자료에 단일선택 혹은 여러선택단추를 추가한다. 여기서 선택된 게시물자료를 삭제한다. 즉

out.print(“<p align=left><input type=checkbox name=D” +id+ “value=del>” +id+ “번째 게시물을 삭제합니다”);

while순환명령문을 리용하여 자료표에서 매 마당의 내용을 읽어들인다. 그리고 우에서와 같이 단일선택 및 여러선택단추를 추가한다. 만일 관리자가 《삭제확정》단추를 찰칵하면 선택된 자료를 delete.jsp에 보내여 삭제조작을 진행한다.

실례 8-12

delete.jsp

```
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@include file="opendata.jsp"%>
<%
    String para,value;
    int pageno,numf,numl;
    if(request.getParameter("pageno")==null)
        pageno=0;
    else
        pageno=Integer.parseInt(request.getParameter("pageno"));
        numf=pageno*5-4;
        numl=numf+4;
        for(int i=numf;i<=numl;i++)
        {
            para="D"+String.valueOf(i);
            value=request.getParameter(para);
            if(value!=null)
            {
                    sql="delete from message where id="+i;
                    smt.executeQuery(sql);
            }
        }
    sql="alter table message drop id";
    smt.executeQuery(sql);
    response.sendRedirect("manager.jsp");
%>
```

프로그람설명

① 우에서의 프로그람 manager.jsp는 관리자가 있는 다음페지 pageno를 이 프로그람에 보낼수 있다. 왜냐하면 관리자가 삭제한 게시물이 반드시 이 페지에 있어야 하기때문이다. 다음에 pageno를 리용하여 이 페지의 numf와 numl을 계산한다. 순환 《 for(int i=numf;i<=numl;i++)》은 《Di》가 《null》과 같은가 같지 않는가를 검사하는데 쓰인다. 같지 않으면 이 내용이 《del》이다는것을 표시하며 다음에 자료표로부터 등록번호가 i인 자료를 삭제한다.

례제를 하나 보기로 하자. 만일 관리자가 5번째 페지에서 23, 24번째 항목의 게시물을 선

택하여 삭제를 진행한다면 《D23= “del”》, 《D24= “del”》이다. 이 프로그람의 for순환식은 《for(int i=21;i<=25;i++)》으로서 《D21~D25》에서의 값이 《null》과 같은가를 검사하며 같지 않으면 이 등록번호의 자료를 삭제한다.

② 자료표에서 몇개의 게시물을 삭제한 다음에 반드시 자료표에 대하여 다시 새롭게 정렬하는 작업을 하여야 한다. 때문에 자료삭제 후에 매 항목자료의 id는 앞으로 자동보충되며 페지가르기의 열람방식은 id를 통하여 완성된다. 우의 프로그람에서는 삭제한 다음 id들이 자동정렬되도록 코드가 작성되였다. 집행결과는 그림 8-5에서 보여준다.

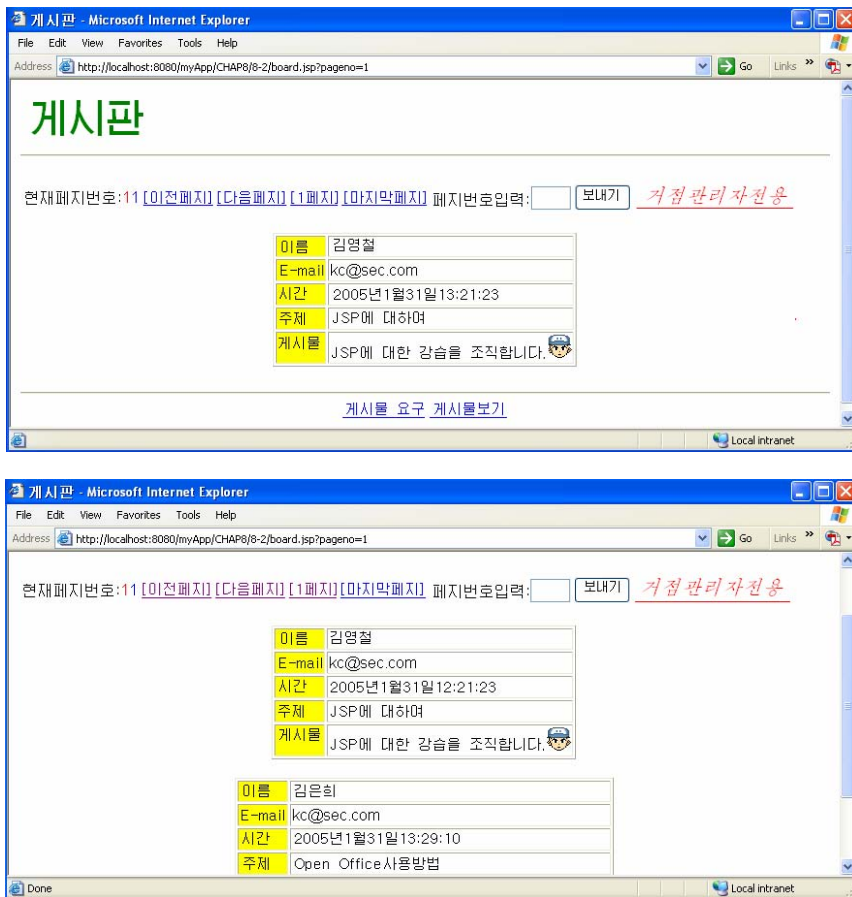



그림 8-5. board.jsp의 현시화면

계속해서 이 게시판프로그람의 페지가르기열람기능을 그림 8-6에서 보여준다.

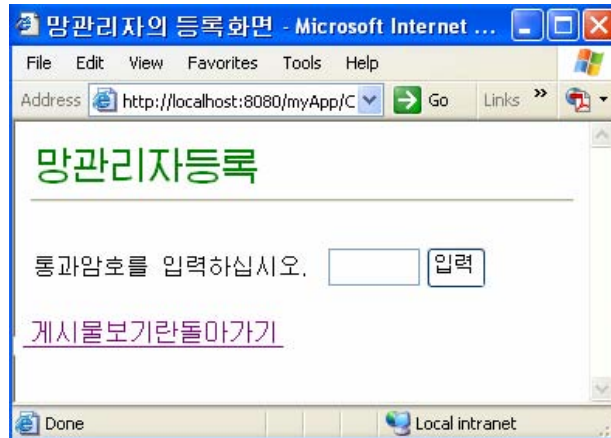



그림 8-6. Manager.jsp의 현시화면

제3절. 간단한 대화실의 작성

망상에서의 대화는 이미 콤퓨터사용자들속에서 하나의 흐름으로 되고있다. 망열람자들은 망을 통해 여러곳에 있는 사람들과 교류하고 정보를 전달한다. 심지어 일부 사람들이 즐거운 시간을 보낼수 있게 하는 정보의 전달과 오락들이 망봉사를 통해 진행되고있다.

이 절에서는 간단한 대화실을 작성해 보기로 한다. 이 대화실에서는 사용자의 IP주소를 현시할수 있으며 대화실정보의 갱신방식 역시 사용자가 선택하여 수동 또는 자동 갱신할수 있다. 아래에서는 이 대화실의 파일구성과 대체적인 기능을 소개한다.

- enter.jsp: 대화실에 들어가는 등록화면이다.
- check.jsp: 사용자의 입력을 검사한다.
- chatroom.htm: 대화실의 틀홈페지이다.
- opendata.jsp: MySQL자료기지와 접속하는데 쓰이는 프로그람이다.(실례 8-3참고)

- words.jsp: 발신 및 발신내용을 처리하는 프로그람이다.
- saveword.jsp: 사용자가 발신한 내용을 보존하는 프로그람이다.
- view.jsp: 가장 최근의 발신내용을 현시한다.

이 대화실에는 chatroom자료표가 사용된다. 내용은 아래와 같다. 즉

```
mysql>create table chatroom(
        ->nick char(20),
        ->sex char(10),
        ->color char(10),
        ->time char(30),
        ->note text,
        ->id int not null auto_increment,
        ->primary key(id)
        ->);
```

이 자료표의 《nick》마당은 사용자가 대화실에 등록할 때의 새김을 기록하는데 쓰인다. 《sex》마당은 사용자성별에 따르는 아이콘파일이름을 기록한다. 《color》마당은 사용자가 선택한 색갈을 기록한다. 《time》마당은 발신시간을 기록한다. 《note》마당은 발신내용을 기록한다. 《id》는 자동적으로 하나 증가할수 있는 주색인마당이다.

이제는 완성된 프로그람에 대해서 보도록 하자.

실례 8-13

enter.jsp

```
<HTML>
<TITLE>대화실들어가기</TITLE>
<BODY>
<font color=green size=5>등록화면</font>
<form action=check.jsp method=Post>
<table border=1>
   <tr>
      <td bgcolor=yellow>대화자</td><td><input type=text size=20 name=nick></td>
   </tr>
   <tr>
      <td bgcolor=yellow>성별</td><td>남
      <input type=radio name=sex value=boy.gif checked>
      녀<input type=radio name=sex value=girl.gif></td>
   </tr>
```

```
    <tr>
        <td bgcolor=yellow>색갈</td>
        <td><select size=1 name=color>
        <option selected>무색
        <option>적색
        <option>황색
        <option>청색
        <option>록색
        </select></td>
    </tr>
    <tr>
        <td bgcolor=yellow>갱신방식</td>
        <td><select size=1 name=change>
        <option selected>수동갱신
        <option>5초마다 갱신
        <option>10초마다 갱신
        <option>30초마다 갱신
        </select></td>
    </tr>
    <tr align=center><td colspan=2>
        <input type=submit name=send value=대화실들어가기>
        <input type=reset value=다시입력></td>
    </tr>
</table>
</form>
<font color=red>
<%
   String errmsg=request.getParameter("errmsg");
   if(errmsg!=null)
      out.print(errmsg);
%>
</font>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 대화실의 등록화면을 현시하는 프로그람이다.

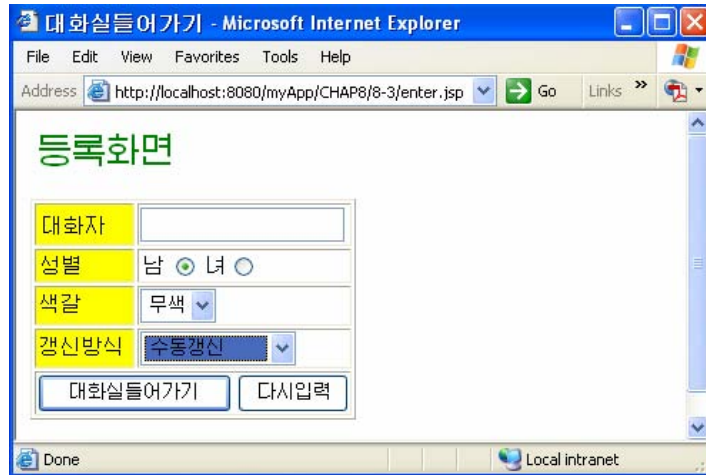



그림 8-7. 대화실의 등록화면

이 홈에서 사용자는 성별, 좋아하는 색갈, 대화실의 갱신방식을 선택할수 있고 사용자가 새김을 입력하지 않았다면 아래의 check.jsp를 통하여 검사한 후 오유정보를 보낸다.

실례 8-14

check.jsp

```
<%
request.getSession(true);
if(request.getParameter("nick").length()==0)
{
        String errmsg="새김을 입력하십시오!!";
        response.sendRedirect("enter.jsp?errmsg="+errmsg);
}
else
{
        String color=request.getParameter("color");
        String change=request.getParameter("change");
        String nick=request.getParameter("nick");
        String sex=request.getParameter("sex");
        if(color.equals("무색"))
                color="black";
        else if(color.equals("적색"))
                color="red";
        else if(color.equals("황색"))
```

```
                color="yellow";
        else if(color.equals("청색"))
                color="blue";
        else
                color="green";
        if(change.equals("수동갱신"))
                change="0";
        else if(change.equals("5초마다 갱신"))
                change="5";
        else if(change.equals("10초마다 갱신"))
                change="10";
        else
                change="30";
        session.putValue("color",color);
        session.putValue("change",change);
        session.putValue("nick",nick);
        session.putValue("sex",sex);
        response.sendRedirect("chatroom.htm");
}
%>
```

프로그람설명

① 먼저 사용자대화접속을 작성하고 여기에 사용자자료를 추가한다. 계속하여 사용자가 새김을 입력하였는가 아닌가를 판단하여 입력하지 않았으면 오유정보 《errmsg》를 enter.jsp에 보낸다.

② if…else if…else판단명령문을 사용하여 사용자가 선택한 색갈에 대응하는 수값으로 전환하여 color변수에 넣는다.

③ if…else if…else판단명령문을 사용하여 사용자가 선택한 갱신방식에 대응하는 수값으로 전환하여 change변수에 넣는다. 만일 사용자가 수동갱신을 선택하면 《change= "0"》이라는데 주의를 돌리시오.

④ color, change, nick, sex 등의 정보를 사용자대화접속에 넣는다. 다음에 《response.sendRedirect ("chatroom.htm")》을 통해 대화실의 주페지에 들어간다.

실례 8-15

chatroom.htm

```
<HTML>
<TITLE>대화실</TITLE>
<frameset rows="30%,70%">
<frame src=words.jsp name=words>
<frame src=view.jsp name=view>
</frameset>
</HTML>
```

프로그람설명

이 HTML파일은 그림 8-8과 같은 대화실들을 만드는데 리용한다.

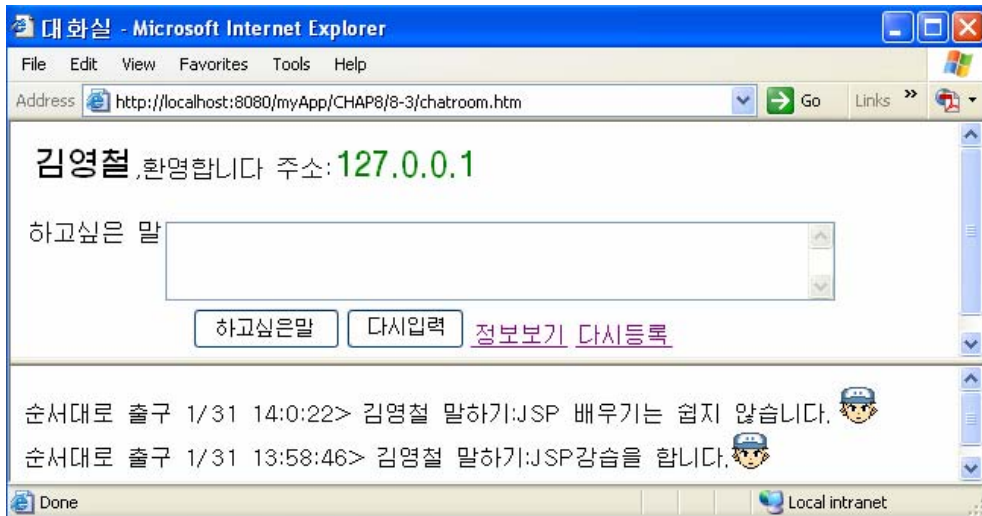



그림 8-8. chatroom.htm페지

실례 8-16

Words.jsp

```
<HTML>
<BODY>
<base target=view>
<font color=<%=session.getValue("color")%>size=+2>
<%=session.getValue("nick")%></font>,환영합니다
주소:<font color=green size=+2><%=request.getRemoteAdd()%></font><font
color=red>
```

```
<%
  String errmsg=request.getParameter("errmsg");
  if(errmsg!=null)
  out.print(errmsg);
%>
</font>
<form action=saveword.jsp method=Post>
<table border=0>
    <tr>
        <td valign=top>하고싶은 말</font></td>
        <td valign=top><textarea name=note rows=2 cols=60></textarea></td>
    </tr>
    <tr>
        <td colspan=2><center><input type=submit name=send value=하고싶은말>
        <input type=reset name=clear value=다시입력>
        <a href=view.jsp>정보보기</a>
        <a href=enter.jsp target=_top>다시등록</a>
    </tr>
</table>
</form>
</BODY>
</HTML>
```

프로그람설명

① 이 페지는 하나의 홈을 제공하고있으며 여기에 사용자는 발신내용을 입력할수 있다. 이 홈의《form action=saveword.jsp》은 사용자가 정보를 발송한 다음에는 saveword.jsp에 의하여 처리되도록 하는 코드이다.

②《정보보기》는 페지에 있는《정보보기》를 찰칵하면 view.jsp 페지가 현시되여 saveword.jsp에서 얻어진 결과가 현시되도록 한다.

《다시등록》은 페지에서 《다시등록》을 찰칵하면 《enter.jsp》가 현시되여 새로운 사용자를 등록할수 있도록 한다.

③ 《request.getRemoteAddr()》를 사용하여 사용자의 IP주소를 얻는다.

실례 8-17

saveword.jsp

```
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
String note=request.getParameter("note");
if(note.length()==0)
{
        String errmsg="정보가 없는것은 입력할수 없습니다!!";
        response.sendRedirect("view.jsp?errmsg="+errmsg);
}
else
{
int month,day,hour,minute,second;
        String nick=(String)session.getValue("nick");
        String sex=(String)session.getValue("sex");
        String color=(String)session.getValue("color");
        GregorianCalendar calendar;
        calendar=new GregorianCalendar();
        month=calendar.get(Calendar.MONTH)+1;
        day=calendar.get(Calendar.DAY_OF_MONTH);
        hour=calendar.get(Calendar.HOUR_OF_DAY);
        minute=calendar.get(Calendar.MINUTE);
        second=calendar.get(Calendar.SECOND);
        note=convert(note);
        nick=convert(nick);
        String time=month+"/"+day+" "+hour+":"+minute+":"+second;
        sql="insert into chatroom(nick,sex,color,time,note) values
('"+nick+"','"+sex+"','"+color+"','"+time+"','"+note+"')";
        smt.executeQuery(sql);
        response.sendRedirect("view.jsp");
}
%>
```

프로그람설명

우선 사용자가 발송한 정보내용이 있는가 없는가를 판단한다. 만일 없으면 오유정보를 view.jsp에 보낸다. 내용이 있으면 현재시간을 시간단위로 얻어 변수 《time》에 넣는다. 그 다음 사용자자료와 발언내용을 chatroom자료표에 함께 넣는다.

여기서 주의할것은 사용자새김과 발언을 의미하는 《nick》와 《note》변수는 반드시 먼저 convert자체정의함수를 사용하여 변환해주어야 정확히 조선글을 현시할수 있다.

실례 8-18

view.jsp

```
<HTML>
<BODY>
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%@page contentType="text.html;charset=Big5"%>
<%
String errmsg=request.getParameter("errmsg");
if(errmsg!=null)
{
	errmsg=convert(errmsg);
out.print("<font color=red size=5>"+errmsg+"</font>");
}
else {
	String nick,sex,color,time,note;
	String refresh=(String)session.getValue("change");
	if(!refresh.equals("0"))
		response.setHEADer("Refresh",refresh);
	sql="select * from chatroom order by id desc limit 8";
	rs=smt.executeQuery(sql);
	while(rs.next())
	{
		nick=rs.getString(1);
		sex=rs.getString(2)
		color=rs.getString(3);
		time=rs.getString(4);
		note=rs.getString(5);
		out.print(time+"><font color="+color+">"+nick+"말하기:"+note+"
	</font><img src="+sex+"><BR>");
	}
}
%>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 대화실의 최신정보를 현시하며 사용자의 선택에 따라 수동 또는 자동으로 홈페지내용을 갱신할수 있도록 한다.

① if else판단명령문은 우선 웃페지(saveword.jsp)가 오유정보(errmsg)를 보내는가를 판단한다. 만일 있으면 이것을 정보보기란에 현시하며 그렇지 않으면 대화실의 정보를 현시한다.

② 《String refresh=(String)session.getValue("change")》는 사용자가 선택한 대화실정보의 갱신시간을 얻어 refresh변수에 넣는다. 또한 《if(!refresh.equals("0"))은 refresh의 값이 《0》인가를 판단하며 《0》이면 사용자가 《수동갱신》을 선택하였다는것을 의미하며 그렇지 않으면 《response.setheader("Refresh" ,refresh)》은 refresh의 값을 리용하여 정보보기란이 자동갱신되는 시간간격을 설정한다.

③ 최신발언정보를 현시하기 위하여 SQL명령 《sql= "select * from chatroom order by id desc limit 8" 》을 설정한다. 여기서 《order by id desc》는 id마당이 색인이 큰데서부터 작은데로 배렬된다는것을 의미한다. 발언을 자료표에 넣을 때 id마당이 자동적으로 하나 증가하므로 id마당값이 큰 자료일수록 새로운 자료로 되는것이다. 그리고 "limit 8"은 마지막에서부터 8개의 항목자료들을 얻으므로 최신발언내용들을 읽어낼수 있다.

④ while순환을 리용하여 얻은 8개항목으로 된 최신자료를 대화실의 정보보기란에 현시한다. 아래에서는 이 대화실응용실례의 실행결과를 보여주었다.(그림 8-9)

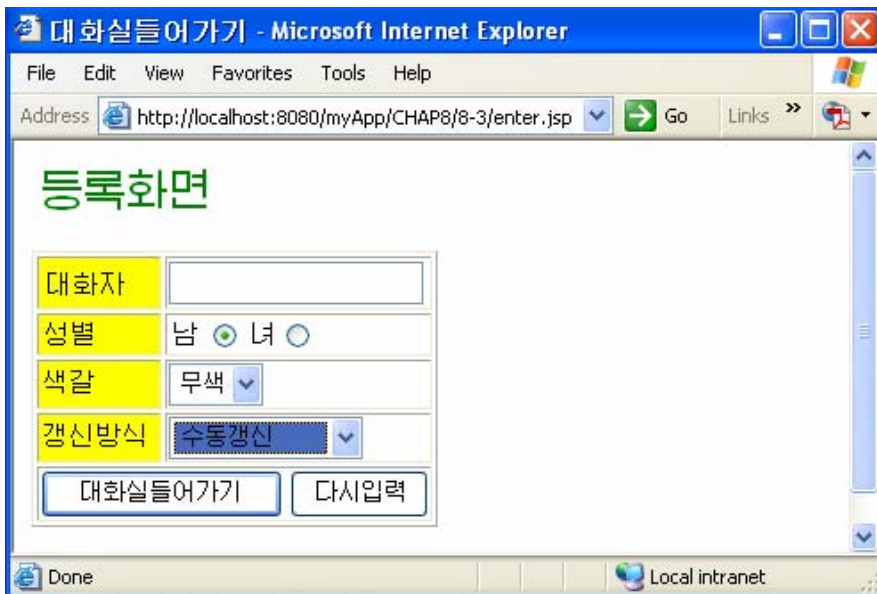



그림 8-9. enter.jsp의 현시화면

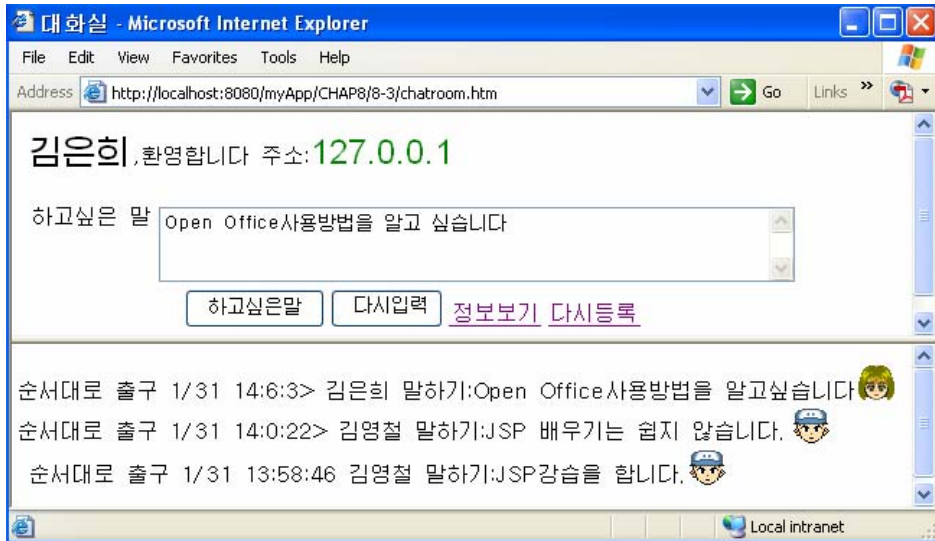



그림 8-10. 대화실에서 대화내용의 게시

사용자는 또한 대화실에서 탈퇴하여 그림 8-10과 같이 새로 다시 등록할수 있다.

제4절. 망연단의 제작

《망연단》페지는 주로 여러 부문의 망열람자들에게 어떤 주제를 주고 기사를 발표할수 있도록 하는 페지이다. 망상에서 이것을 통해 토론하고 교류를 진행할수 있다.

이 절에서는 《망연단》을 어떻게 만드는가를 학습한다. 《망연단》은 아래와 같은 파일들로 구성된다.

- discuss.jsp: 토론의 모든 주제를 현시하는 주프로그람이다.
- mainform.htm: 사용자가 새로운 주제를 입력하는 홈이다.
- opendata.jsp: MySQL자료기지를 열고 접속한다.(실례 8-3참고)
- savemain.jsp: 새로운 토론주제를 자료기지에 넣는다.
- detail.jsp: 기사내용과 이 기사의 회답파일을 현시한다.
- reply.jsp: 사용자가 기사에 회답하는 홈이다.
- savedata.jsp: 회답한 기사를 자료기지에 넣는다.
- convert.jsp: 조선글문자렬과 행바꾸기를 처리하는 자체정의함수파일이다.

이 응용실례는 한개의 자료표 discuss를 사용하여 토론주제와 회답기사를 보존하고있다. 그 내용은 아래와 같다.

```
mysql>create table discuss(
        ->name char(20),
        ->email char(40),
        ->subject char(60),
        ->content text,
        ->time char(40),
        ->reply int(5),
        ->id int not null auto_increment,
        ->primary key(id),
        ->);
```

이 자료표에서 《name》, 《email》, 《subject》, 《content》, 《time》, 《id》 등 마당들은 각각 어떤 주제나 기사의 《발기인》, 《발기인의 전자우편》, 《기사주제》, 《기사내용》, 《발기시간》 및 《기사번호》를 기록하는데 쓰인다.

그리고 reply마당을 사용하여 이 기사가 어느 기사에 회답한것인가를 얻는다. 즉 이 마당은 기사에 대한 회답번호이다. 실례를 들어 어떤 기사가 번호가 19인 기사의 회답파일이라면 이 기사의 reply마당값은 19로 된다. 주제기사에서 이 마당값은 0으로 설정된다. 왜냐하면 주제는 아무런 기사의 회답파일이 아니기때문이다. 이러한 개념을 료해한 다음에는 완전한 응용실례프로그람의 내용을 보도록 하자.

실례 8-19

discuss.jsp

```
<HTML>
<TITLE>직결토론마당</TITLE>
<BODY>
<center><font color=green size=5>직결토론마당-주제보기란에 온것을 환영합니다</font>
<hr>
<table border=1>
    <tr bgcolor=yellow>
    <td>날자시간</td><td>저자</td><td>주제</td>
    </tr>
<%@page import="java.sql.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%
   sql="select * from discuss where reply=0";
   rs=smt.executeQuery(sql);
   String name,email,subject,content,time;
```

```
   int reply,id;
   while(rs.next())
   {
        name=ts.getString(1);
        email=rs.getString(2);
        subject=rsgetString(3);
        content=rs.getString(4);
        time=rs.getString(5);
        reply=rs.getInt(6);
        id=rs.getInt(7);
        out.print("<tr><td>"+time+"</td>");
        out.print("<td>"+name+"</td>");
        out.print("<td><a href=detail.jsp?id="+id+">"+subject+"</a></td>");
        out.print("</tr>");
   }
%>
</table>
<hr>
<a href=mainform.htm>새로운 주제</a>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 기본페지 즉 주제보기란이 있는 페지를 현시하는 프로그람이다. 여기서 기본은 자료표로부터 reply마당이 0인 자료를 추출하는것이다. 다시말하여 주제기사를 추출하는것이다. While순환명령문을 리용하여 주제기사에서 매 마당의 자료를 얻고 그 내용을 표에 현시한다. 여기서 프로그람코드 《out.print(“<td>” + “subject+” <td>”)》은 필요한 마당들에 값들을 넣어(실례로 id마당에) detail.jsp파일에 보낸다. 즉 기사번호를 이 프로그람에 전달한다. 이렇게 하면 detail.jsp는 번호를 통하여 기사내용과 그의 회답파일을 현시한다.

실례 8-20

mainform.htm

```
<HTML>
<TITLE>새로운 주제</TITLE>
<BODY>
<center>
<font color=green size=5>새로운 주제</font>
여기서 새주제를 발기할수 있습니다. 그것은 주제보기란에 가입할수 있다.
```

```
<hr>
<form action=savemain.jsp method=Post>
<table border=1>
   <tr>
      <td bgcolor=yellow>이름</td><td><input type=text size=20 name=name></td>
   </tr>
   <tr>
      <td bgcolor=yellow>E-mail</td><td><input type=text size=40 name=email></td>
   </tr>
   <tr>
      <td bgcolor=yellow>주제</td><td><input type=text size=60 name=subject></td>
   </tr>
   <tr>
      <td valign=top bgcolor=yellow>내용</td>
      <td><textarea name=content rows=4 cols=60></textarea></td>
   </tr>
   <tr align=center>
      <td colspan=2><input type=submit name=send value=새주제보내기>
      <input type=reset value=반복쓰기없애기></td>4
   </tr>
</table>
<hr>
<a href=discuss.jsp>주제보기란돌아가기</a>
</form>
</BODY>
</HTML>
```

프로그람설명

이것은 사용자가 새로운 주제를 발기하는 홈이다. 이 홈의 입력자료는 savemain.jsp에 의하여 처리되고 판단되며 내용은 자료표에 추가된다.

실례 8-21

savemain.jsp

```
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
```

```
String name=request.getParameter("name");
String email=request.getParameter("email");
String subject=request.getParameter("subject");
String content=request.getParameter("content");
if(name.length()==0||email.length()==0||subject.length()==
0|content.length()==0)
{
        String errmsg="마당은 빌수 없습니다!!";
        out.print("<center><font color=green size=5>오유정보
                <hr></font><font color=red>"+errmsg+"</font><hr>");
        out.print("<input type=button value=이전페지돌아가기
                onclick=history.back();>");
}
else
{
        int year,month,day,hour,minute,second;
        String time;
        GregorianCalendar calendar;
        calendar=new GregorianCalendar();
        year=calendar.get(Calendar.YEAR);
        month=calendar.get(Calendar.MONTH)+1;
        day=calendar.get(Calendar.DAY_OF_MONTH);
        hour=calendar.get(Calendar.HOUR_OF_DAY);
        minute=calendar.get(Calendar.MINUTE);
        second=calendar.get(Calendar.SECOND);
        time=year+"년"+month+"월"+day+"일"+hour+":"+minute+":"+second;
        name=convert(name);
        subject=convert(subject);
        content=convert(content);
        content=Replace(content);
        sql="insert into discuss(name,email,subject,content,time,reply)values
               ('"+name+"','"+email+"','"+content+"','"+time+"',0)";
        smt.executeQuery(sql);
        response.sendRedirect("discuss.jsp");
}
%>
<HTML>
<TITLE>오유정보</TITLE>
</HTML>
```

프로그람설명

우선 사용자가 모든 마당에 자료를 입력하였는가 하지 못했는가를 판단한다. 만일 어느 한 마당이라도 내용이 없으면 오유정보를 현시하고 그렇지 않으면 현재 시간을 시간단위로 얻어 time변수에 넣는다. 그다음 time변수의 값을 매 마당의 입력자료와 함께 자료표에 넣는다. 만일 매 자료변수의 내용이 조선글자나 행바꾸기부호를 포함한다면 반드시 자체정의함수를 사용하여 전환하여야 한다.

마지막으로 《response.sendRedirect(“discuss.jsp”)》를 리용하여 주제보기란에로 다시 돌아와 새로운 주제를 렬거한다.

실례 8-22

detail.jsp

```
<%@page import="java.sql.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%
  int id=Integer.parseInt(request.getParameter("id"));
  sql=="select * from discuss where id="+id;
  rs=smt.executeQuery(sql);
  String name=rs.getString(1);
  String email=rs.getString(2);
  String subject=rs.getString(3);
  String content=rs.getString(4);
  String time=rs.getSring(5);
  int back=rs.getInt(6);
  id=rs.getInt(7);
%>
<HTML>
<TITLE>파일보기란</TITLE>
<BODY>
<center><font color=green size=5>직결토론마당-파일보기란에 들어온것을
환영합니다</font>
<hr>
<table border=1>
   <tr>
       <td bgcolor=yellow>저자</td><td><%=name%></td>
   </tr>
   <tr>
       <td bgcolor=yellow>E-mail</td><td><a
       href=mailto:<%=email%>><%=email%></td>
   </tr>
```

```
    <tr>
        <td bgcolor=yellow>주제:</td><td><%=subject%></td>
    </tr>
    <tr>
        <td bgcolor=yellow>시간:</td><td><%=time%></td>
    </tr>
    <tr>
        <td bgcolor=yellow>내용:</td>,td><%=content%></td>
    </tr>
</table>
<hr>
<font size=4 color=green>이 기사의 파일에 회답</font><p>
<table border=0>
<%
  sql="select * from discuss where reply="+id;
  rs=smt.executeQuery(sql);
  while(rs.next()){
        name=rs.getString(1);
        subject=rs.getString(3);
        time=rs.getString(5);
        int select=rs.getInt(7);
        out.print("<tr><td>"+time+"</td>");
        out.print("<td>"+name+"</td>");
        out.print("<td><a href=detail.jsp?id="+select+">"+subject+"</a></td>");
        out.print("</tr>");
  }
%>
</table><p>
<a href=reply.jsp?id=<%=id%>>기사회답</a>
<a href=discuss.jsp>주제보기란에로 돌아가기</a>
<%
  if(back==0)
    out.print("<a href=discuss.jsp>이전층으로 가기</a>");
  else
    out.print("<a href=detail.jsp?id="+back+">다음층으로 가기</a>");
%>
</BODY>
</HTML>
```

프로그람설명

실행결과를 보여주는 그림 8-11, 8-12를 통해 detail.jsp를 설명한다.

① 주프로그람(discuss.jsp)에서 표의 《 주제 》렬의 임의의 주제내용을 선택하면 detail.jsp프로그람이 실행된다. 계속하여 자료표에서 id마당의 값(기사번호)에 대응하는 자료를 찾아 파일보기란에 현시한다.

② 이 기사의 내용을 현시한 다음 프로그람은 다시 자료표에서 이 기사에 회답하는 파일을 찾는다. 즉 reply마당을 리용하여 이 기사번호에 해당한 모든 자료들을 질의문 《sql= "select * from discuss where reply=" +id》에 의하여 찾는다. 다음 《int select=rs.getInt(7)》을 리용하여 변수 《select》에 자료의 번호(id)를 대입하고 이 번호에 해당한 회답파일을 현시한다.(그림 8-12)

③ 파일보기란 아래에는 하이퍼련결기호가 붙은 3개의 본문이 있다. 그것들은 각각《기사회답》, 《주제보기란에로 돌아가기》, 《이전층으로 가기》이다. 《기사회답》에서는 reply.jsp를 동작시켜 이 기사의 번호 《select》을 보내 이 기사가 어느 기사에 회답되였는가를 기록한다. 《주제보기란에로 돌아가기》는 discuss.jsp페지에로 넘어가도록 한다. 《이전층으로 가기》는 back변수에 의해 이전 기사에 돌아가거나 주제보기란으로 돌아가도록 한다.

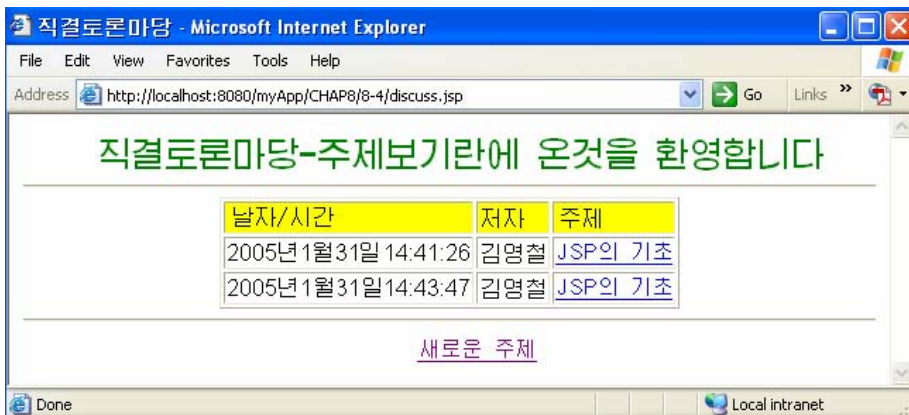



그림 8-11. discuss.jsp의 현시결과

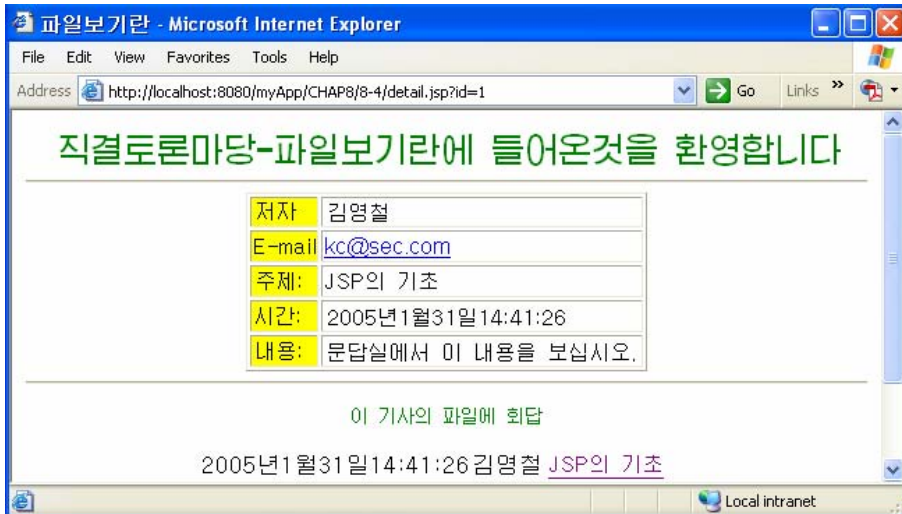



그림 8-12. detail.Jsp의 현시결과

실례 8-23

reply.jsp

```
<%
String id=request.getParameter("id");
%>
<HTML>
<TITLE>파일회답</TITLE>
<BODY>
<center>
<font size=5 color=green>파일회답</font>
여기서 회답하여 회답파일에 추가할수 있습니다.
<hr>
<form action=savedata.jsp?id=<%=id%>method=Post>
<table border=1>
   <tr>
      <td bgcolor=yellow>이름</td>
      <td><input type=text size=20 name=name></td>
   </tr>
   <tr>
      <td bgcolor=yellow>E-mail</td>
      <td><input type=text size=40 name=email></td>
   </tr>
   <tr>
      <td bgcolor=yellow>주제</td>
      <td><input type=text size=60 name=subject></td>
```

```
    </tr>
    <tr>
        <td valign=top bgcolor=yellow>내용</td>
        <td><textarea name=content rows=4 cols60></textarea></td>
    </tr>
    <tr>
        <td align=center colspan=2><input type=submit name=send value=기사보내
        기>
        <input type=reset value=다시쓰기없애기></td>
    </tr>
</table>
<hr>
<a href=discuss.jsp>주제보기란가기</a>
<a href=detail.jsp?id=<%=id%>>이전페지로 가기</a>
</form>
</BODY>
</HTML>
```

프로그람설명

이 프로그람은 기사에 회답하여 이 홈의 내용을 savedata.jsp에서 처리하게 한다. 첫번째 행을 통해 읽어들인 id번호를 통해 프로그람이 어느 기사에 회답하는가를 기록하거나 이전페지에로 돌아갈수 있다.

실례 8-24

savedata.jsp

```
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
String name=request.getParameter("name");
String email=request.getParameter("email");
String subject=request.getParameter("subject");
String content=request.getParameter("content");
String back=request.getParameter("id");
if(name.length(0==0||email.length()==0
                ||subject.length()==0||content.length()==0)
{   String errmsg="마당은 빌수 없습니다!!";
```

```
    out.print("<center><font color=green size=5>오유정보<hr></font>
              <font color=red>"+errmsg+"</font><hr>");
    out.print("<input type=button value=이전페지가기 onclick=history.back()>");
}
else
{
        int year,month,day,hour,minute,second;
        String time;
        GregoianCalendar calendar;
        calendar=new Gregoiancalendar();
        year=calendar.get(Calendar.YEAR);
        month=calendar.get(Calendar.MONTH)+1;
        day=calendar.get(Calendar.DAY_OF_MONTH);
        hour=calendar.get(Calendar.HOUR_OF_DAY);
        minute=calendar.get(calendar.MINUTE);
        second=calendar.get(Calendar.SECOND);
        time=year+"넌"+month+"월"+day+"일"+hour+":"+minute+":"+second;
        name=convert(name);
        subject=convert(subject);
        content=convert(content);
        content=Replace(content);
       sql="insert into discuss(name,email,subject,content,time,reply)
                values('"+name+"','"+email+"','"subject+"',
                '"+content+"','"+time+"','"+back+"')";
        smt.executeQuery(sql);
        response.sendRedirect("detail.jsp?id="+back);
}
%>
<HTML>
<TITLE>오유정보</TITLE>
</HTML>
```

프로그람설명

이 프로그람은 망열람자의 회답파일을 자료기지에 넣는다. 그리고 주제를 보존하는 savemain.jsp와 류사하게 우선 매 마당의 내용이 비였는가 아닌가를 판단하고 현재의 시간문자렬을 얻는다. 아래에서는 완전한 《망연단》응용실례프로그람의 집행결과를 보여준다.(그림 8-13)

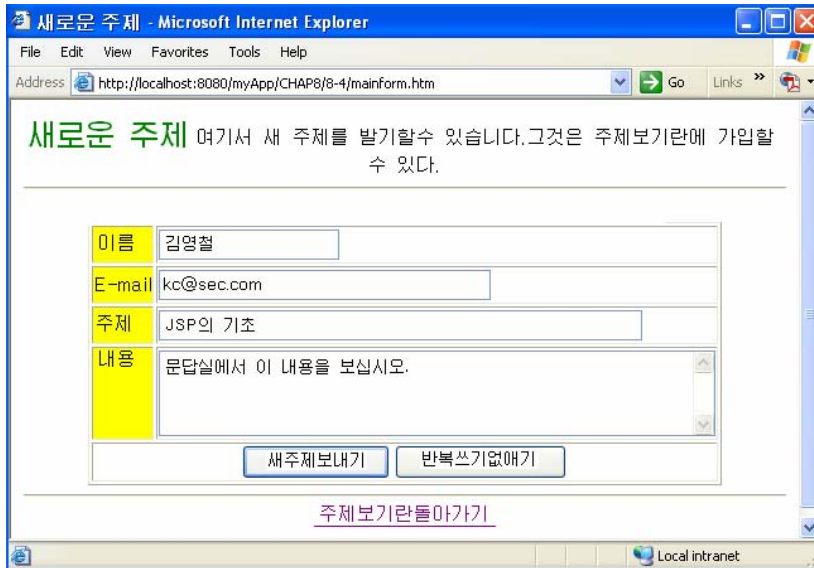



그림 8-13. mainform.htm의 현시결과

제5절. 가입자관리홈페지의 작성

가입자관리홈페지기능은 현재의 거점구축에서 아주 중요하며 특히는 료금을 물고 가입자인증을 얻는 회사들에서 중요하게 제기된다. 불법사용자나 딴마음을 가진 해커들을 방지하기 위하여 가입자인증 및 관리방식은 반드시 필요한것이다. 이 절에서는 이것을 실현한 가입자관리홈페지를 만들어 보기로 한다.

여기서 만드는 《가입자관리홈페지》는 새로운 가입자의 등록, 통과암호의 탐색, 개인자료의 수정, 거점관리의 기능을 갖추고있다. 이 실례는 비교적 많은 파일들을 사용하고있다.(표 8-3)

표 8-3. 가입자관리홈페지에 속하는 프로그람들

프로그람	설 명
main.htm	이 프로그람의 주페지
title.htm	홈페지상에서 표제를 현시
select.htm	사용자가 임의의 페지와의 련결을 선택할수 있도록 하는 페지
newuser.htm	새 사용자를 등록하는 홈
opendata.jsp	자료기지를 여는 프로그람
savedata.jsp	새 가입자등록을 처리하는 프로그람
login.jsp	사용자등록기능을 처리하는 프로그람
member.jsp	가입자인가 아닌가를 검증할수 있는 가입자페지
forget.jsp	사용자가 자료를 입력하여 통과암호를 찾는 프로그람

search.jsp	사용자의 통과암호 찾기
modify.jsp	사용자가 자기 자료를 수정할수 있는 홈
update.jsp	사용자가 수정한 자료를 처리하는 프로그람
manager.jsp	관리자의 전용프로그람
delete.jsp	관리자가 자료를 삭제하는 프로그람
convert.jsp	조선글문자렬과 행바꾸기를 처리하는 자체정의함수파일

아래에서 이 프로그람들사이의 련관구조를 보여준다.(그림 8-14)

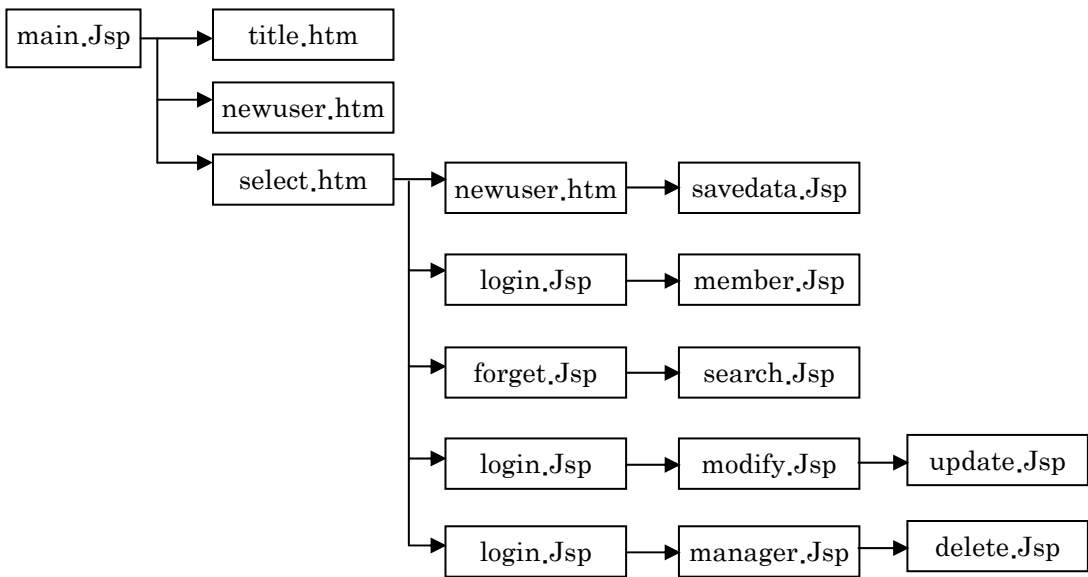



그림 8-14. 프로그람들사이의 련관구조

프로그람설명

또한 이 실례에서는 한개의 자료표로 매 가입자의 자료를 관리하는데 이 자료표이름은 personal이다.

```
mysql>create table personal(
      ->name char(20),
      ->email char(40),
      ->address char(60),
      ->password char(10),
      ->sex char(8),
      ->id int not null auto_increment,
      ->primary key(id)
      ->);
```

자료표의 《name》, 《email》, 《address》, 《password》, 《sex》 등의 마당들은 각각 매 가입자의 이름, 전자우편주소, 주소, 통과암호와 성별을 기록하는데 쓰이며 《id》는 매가입자자료의 등록번호이다.

아래에서 이 실례의 완전한 프로그람을 보도록 하자.

실례 8-25

main.htm
```
<HTML>
<TITLE>가입자관리홈페지</TITLE>
<frameset rows="15%,80%">
<frame src=title.htm name=title>
<frameset cols="15%,75%">
<frame src=select.htm name=select>
<frame src=newuser.htm name=show>
</frameset>
</frameset>
</HTML>
```

프로그람설명

이 HTML원천코드는 가입자관리홈페지의 주페지이다.(그림 8-15)

여기서는 newuser.htm을 열어 마당에 따르는 내용을 모두 입력한 사용자를 새로운 가입자로 등록할수 있게 한다. 만일 사용자가 가입자나 거점관리자이면 왼쪽부분에 있는 하이퍼련결기호가 붙은 임의의 본문을 선택하여 련관된 기능을 사용할수 있다.

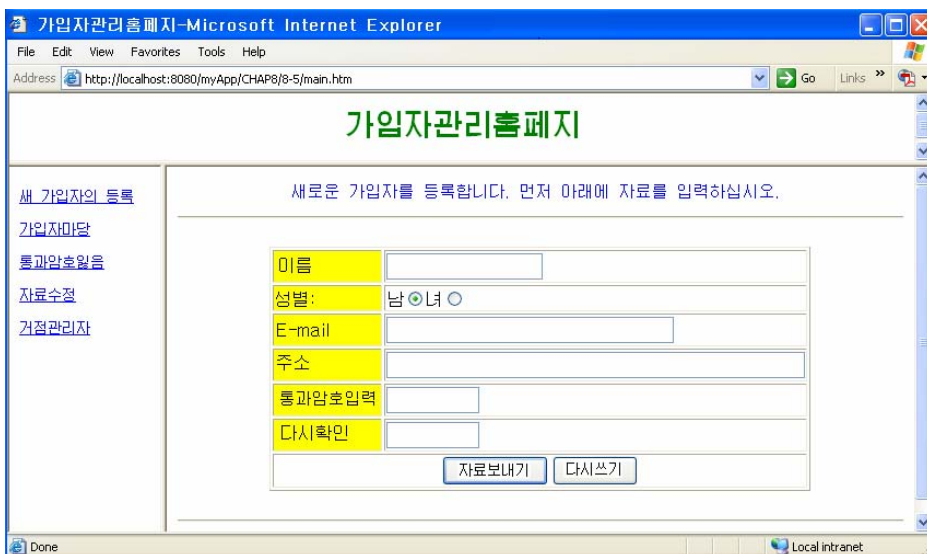



그림 8-15. main.htm의 현시결과

실례 8-26

```
title.htm
<HTML>
<BODY>
<center>
<font color=green size=6><b>가입자관리홈페지</b></font>
</BODY>
</HTML>
```

실례 8-27

```
select.htm
<HTML>
<BODY>
<base target=show>
<a href=newuser.htm>새가입자의 등록</a><p>
<a href=login.jsp?login=1>가입자마당</a><p>
<a href=forget.jsp>통과암호잃음</a><p>
<a href=login.jsp?login=2>자료수정</a><p>
<a href=login.jsp?login=3>거점관리자</a><p>
</BODY>
</HTML>
```

프로그람설명

만일 사용자가 선택한 하이퍼련결본문이 등록가입과 관련되여 있으면 이때의 처리는 프로그람 《login.jsp》가 하게 된다. 여기서 등록번호의 값 login=1, login=2, login=3들을 전달할수 있는데 이 값들에 따라 해당한 처리를 하게 된다. 실례로 login=1이면 가입자마당으로 들어가게 되며 login=2이면 자료수정을 하게 된다.

실례 8-28

```
newuser.htm
<HTML>
<BODY>
<center>
<font color=blue size=4>새로운 가입자를 등록합니다. 먼저 아래에 자료를 입력하십시
오.</font>
```

```
<hr>
<center>
<form action=savedata.jsp method=Post>
<table border=1>
   <tr>
      <td bgcolor=yellow>이름</td><td><input type=text size=20 name=name></td>
   </tr>
   <tr>
      <td bgcolor=yellow>성별:</td><td>남<input type=radio name=sex value=boy
      checked>녀<input type=radio name=sex value=girl></td>
   </tr>
   <tr>
      <td bgcolor=yellow>E-mail</td><td><input type=text size=40 name=email></td>
   </tr>
   <tr>
      <td bgcolor=yellow>주소</td><td><input type=text size=60 name=address></td>
   </tr>
   <tr>
      <td bgcolor=yellow>통과암호입력</td><td><input type=password size=10
      name=password></td>
   </tr>
   <tr>
      <td valign=top bgcolor=yellow>다시확인</td><td><input type=password size=10
      name=confirm></td>
   </tr>
   <tr align=center><td colspan=2>
      <input type=submit name=send value=자료보내기>
      <input type=reset value=다시쓰기></td>
   </tr>
</table>
</form>
<hr>
</BODY>
</HTML>
```

프로그람설명

이 HTML파일은 사용자입력홈으로서 망열람하려는 사용자가 자료를 입력하여 새 가입자로 등록할수 있게 한다.

실례 8-29

savedata.jsp

```
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%!
  boolean checkemail(String str)
  {
        int index;
        boolean check;
        if((index=str.indexOf("@"))!=-1)
                check=true;
        else
                check=false;
        return(check);
  }
%>
<%
 String name=request.getParameter("name");
 String sex=request.getParameter("sex");
 String address=request.getParameter("address");
 String email=request.getParameter("email");
 boolean check=checkemail(email);
 String password=request.getParameter("password");
 String confirm=request.getParameter("confirm");
 sql="select * from personal where name='"+name+"'";
 rs=smt.executeQuery(sql);
 out.print("<center>");
 if(name.length()==0)
 {
        String errmsg="ID마당은 빌수 없습니다!");
        out.print("<font color=green size=5>오유정보<hr></font><font
 color=red>"+errmsg+"</font><hr>");
 }
 else if(rs.getString(1)!=null)
 {
        String errmsg="이 등록번호는 이미 어떤 사람이 사용하고 있습니다.
                새로운 ID로 바꾸십시오!";
        out.print("<font color=green size=5>오유정보<hr></font><font
 color=red>"+errmsg+"</font><hr>");
 }
```

```
else if(check==false)
{
       String errmsg="전자우편주소오유!";
       out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else if(address.length()==0)
{
       String errmsg="주소마당은 빌수 없습니다!";
       out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else if(password.length()>10||password.length()<6)
{
       String errmsg="6~10개 문자의 통과암호를 입력하여야 합니다!";
       out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else if(!password.equals(confirm))
{
       String errmsg="통과암호를 다시 확인하십시오!";
       out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else
{
       String errmsg="축하합니다. 당신은 이 홈페지의 가입자로 되였습니다!";
       name=convert(name);
       address=convert(address);
       sql="insert into personal(name,email,address,password,sex)
values('"+name+"','"+email+"','"+address+"','"+password+"','"+sex+"')";
       smt.executeQuery(sql);
       out.print("<center><font color=green size=5>"+errmsg+"</font><hr>");
}
out.print("<input type=button value=이전페지로 가기 onclick=history.back();>");
%>
```

프로그람설명

이 프로그람은 사용자가 매 마당의 자료를 정확히 입력하였는가를 순차적으로 검사한다. 이때 자료표에서 이미 등록된 가입자인가 아닌가를 검사한다. 만일 자료가 정확하지 않거나 중복되면 이에 대한 오유정보를 내보낸다.

① 자체정의함수 checkemail()은 사용자가 입력한 전자우편주소가 《@》부호를 포함하는가를 검사하여 론리형값 《true》 또는 《false》를 되돌려준다.

② 다음 if판단명령문은 매 마당의 입력이 정확한가를 검사하며 사용자가 입력한 자료들이 다른 가입자가 등록한 자료와 중복되지 않는가를 검사한다. 만일 검사가 성공이면 새로운 가입자자료를 자료기지에 추가한다. 그렇지 않으면 오유정보를 보낸다.

③ 통과암호의 검사부분에서 사용자가 입력한 통과암호와 확인은 각각 《password》와 《confirm》을 리용한다. 만일 이 두 변수가 같지 않으면 전후 입력한 통과암호가 불일치하다는것을 의미하며 오유정보를 내보낸다. 이밖에 통과암호의 길이에 대하여 제한을 준다.《else if(password.length()>10|| password.length()<6)》은 사용자가 입력한 통과암호문자가 6~10개 문자사이에 있는가를 검사한다.

실례 8-30

login.jsp

```
<%
String login=request.getParameter("login");
String errmsg=request.getParameter("errmsg");
request.getSession(true);
if(login.equals("1"))
{
        String title="가입자홈페지에 들어갑니다-우선 이름과 통과암호를 입력하십시오
";
        String action="member.jsp";
        session.putValue("title",title);
        session.putValue("action",action);
}
else if(login.equals("2"))
{
        String title="개인자료의 수정-우선 이름과 통과암호를 입력하십시오";
        String action="modify.jsp";
        session.putValue("title",title);
        session.putValue("action",action);
}
else if(login.equals("3"))
{
        String title="거점관리자등록-관리자등록번호와 통과암호를 입력하십시오";
        String action="manager.jsp";
        session.putValue("title",title);
        session.putValue("action",action);
```

```
}
%>
<HTML>
<BODY>
<center>
<font color=blue size=4><%=session.getValue("title")%></font>
<hr>
<form action=<%=session.getValue("action")%> method=Post>
<table border=1>
  <tr>
    <td bgcolor=yellow>이름입력</td><td><input type=text size=20
    name=name></td>
  </tr>
  <tr>
    <td bgcolor=yellow>통과암호입력</td><td>
    <input type=password size=10 name=password></td>
  </tr>
  <tr align=center><td colspan=2>
    <input type=submit name=send value=등록>
    <input type=reset value=다시입력></td>
  </tr>
</table>
</form>
<hr>
<center>
<font color=red size=4>
<%
 if(errmsg!=null && errmsg.equals("err1"))
  out.print("이름이 잘못 입력되였거나 새 가입자로 등록하지 못하였습니다!!";
 else if(errmsg!=null && errmsg.equals("err2"))
  out.print("입력한 통과암호가 정확하지 않습니다!!");
 else if(errmsg!=null && errmsg.equals("err3"))
 out.print("관리자등록번호와 통과암호를 잘못 입력하였습니다!!";
%>
</font>
</BODY>
</HTML>
```

프로그람설명

이 프로그람(login.jsp)에서 사용자의 등록번호와 통과암호를 입력하기전에 《login》의 값(1, 2, 3)을 지정하여 해당페지에 넘어간다. 이때 이 값에 따라 서로 다른 기능을 수행하게 된다.

① 이 실례에서 사용자등록기능은 《가입자마당》, 《자료수정》, 《거점관리자》를 가질수 있으며 얻은 《login》의 값에 따라 《title》 및 《action》을 설정하고 사용자대화접속에 넣는다. 여기서 《title》은 등록하여야 할 표제이며 《action》은 집행하여야 할 프로그람파일이름이다.

② 그 다음에 있는 HTML원천코드는 하나의 홈을 만들어 사용자가 이름과 통과암호를 입력하게 한다.

③ 마지막의 프로그람코드는 오유코드 《err1》, 《err2》, 《err3》에 따라 서로 다른 오유정보를 현시하도록 한다.

실례 8-31

member.jsp

```
<%@page import ="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
  String name=request.getParameter("name");
  name=convert(name);
  String confirm=request.getParameter("password");
  sql="select * from personal where name='"+name+"'";
  rs=smt.executeQuery(sql);
  String user=rs.getString(1);
  String password=rs.getString(4);
  if(user==null){
        String errmsg="err1";
        response.sendRedirect("login.jsp?errmsg="+errmsg+"&login=1");
  }
  else if(!password.equals(confirm))
  {
        string errmsg="err2";
        response.sendRedirect("login.jsp?errmsg="+errmsg+"&login=1");
  }
%>
<HTML>
<BODY>
<center>
<font color=red size=5><%=user%></font>,가입자페지에 온것을 환영합니다.
</BODY>
</HTML>
```

프로그람설명

① 이 프로그람은 가입자전용페지로서 내용은 자체로 만들어 <HTML>와 </HTML>꼬리표사이에 추가할수 있다.

② 제일 앞의 코드는 사용자가 등록할 때 입력한 등록번호와 통과암호가 가입자자료기지에 있는가 없는가를 대비하는데 쓰인다. 만일 존재하지 않으면 등록화면에로 다시 되돌아 가야 한다.

실례 8-32

forget.jsp

```
<HTML>
<BODY>
<center>
<font color=blue size=4>통과암호보기-주소나 전자우편번호를 입력하여 통과암호를 찾
아볼수 있습니다.</font>
<hr>
<form action=search.jsp method=Post>
<table border=1>
  <tr>
     <td bgcolor=yellow>E-mail입력</td><td><input type=text size=40
     name=email></td>
  </tr>
  <tr>
     <td bgcolor=yellow>주소입력</td><td><input type=text size=60
     name=address></td>
  </tr>
  <tr align=center><td colspan=2>
     <input type=submit name=send value=조사>
     <input type=reset value=다시입력></td>
  </tr>
</table>
</form>
<hr>
<center><font color=red>
<%
  String errmsg=request.getParameter("errmsg");
  if(errmsg!=null)
  out.print(errmsg);
%>
</font>
</BODY>
</HTML>
```

프로그램설명

여기서는 사용자가 자기의 전자우편과 주소를 입력하고 두개중 어느 하나가 정확하기만 하면 search.jsp프로그람을 실행하여 통과암호를 찾을수 있다.

실례 8-33

search.jsp

```
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
   String email=request.getParameter("email");
   String address=request.getParameter("address");
   address=convert(address);
   if(email.length()===0 && address.length()==0)
   {
        String errmsg="주소나 전자우편번호를 입력하십시오!";
        response.sendRedirect("forget.jsp?errmsg="+errmsg);
   }
   out.print("<center>");
   if(email.length()!=0)
   {
        sql="select * from personal where email='"+email+"'";
        rs=smt.executeQuery(sql);
        String password=rs.getString(4);
        if(password!=null)
          out.print("<font size=5>당신의 통과암호는 <font color=red>"+password+"</font>");
        else if(address.length()!=0)
        {
               sql="select * from personal where address='"+address+"'";
               rs=smt.executeQuery(sql);
               password=rs.getString(4);
               if(password!=null)
                  out.print("<font size=5>당신의 통과암호는 <font color=red>
                    "+password+"</font>");
               else
               {
                 out.print("<font color=red> 당신의 통과암호를 찾을 방도가
                  없습니다. 제대로 입력하였는가를 검사하십시오!</font><p>");
                 out.print("<input type=button value=다시입력 onclick
                    =history.back();>");
```

```
                }
            }
            else
            {
               out.print("<font color=red> 당신의 통과암호를 찾을 방도가 없습니다.
                  제대로 입력하였는가를 검사하십시오!</font><p>");
               out.print("<input type=button value=다시입력
                                 onclick=history.back();>");
            }
    }
    else if(address.length()!=0)
    {
            sql="select * from personal where address='"+address+"'";
            rs=smt.executeQuery(sql);
            String password=rs.getString(4);
            if(password!=null)
                    out.print("<font size=5>당신의 통과암호는
                            <font color=red>"+password+"</font>");
            else
            {
               out.print("<font color=red> 당신의 통과암호를 찾을 방도가 없습니다.
                        제대로 입력하였는가를 검사하십시오!</font><p>");
               out.print("<input type=button value=다시입력
                        onclick=history.back();>");
            }
    }
%>
```

프로그람설명

우선 사용자가 입력한 전자우편(email)에 따라 조사하는데 성공이면 통과암호를 보내주고 실패하면 다시 주소(address)를 리용하여 조사한다.

실례 8-34

modify.jsp

```
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%
  String name=request.getParameter("name");
  name=convert(name);
  String confirm=request.getParameter("password");
```

```
  sql="select * from personal where name='"+name+"'";
  rs=smt.executeQuery(sql);
  String user=rs.getString(1);
  String email=rs.getString(2);
  String address==rs.getString(3);
  String password=rs.getString(4);
  String sex=rs.getString(5);
  int id=rs.getInt(6);
  if(user==null)
  {
        String errmsg="err1";
        response.sendRedirect("login.jsp?ermsg="+errmsg+"&login=1");
  }
  else if(!password.equals(confirm))
  {
        String errmsg="err2";
        response.sendRedirect("login.jsp?errmsg="+errmsg+"&login=1");
  }
  if(sex.equals("boy"))
  {
        String boy="checked";
        session.putValue("boy",boy);
  }
  else
  {
        String girl="checked";
        session.putValue("girl",girl);
  }
%>
<HTML>
<BODY>
<center>
<font color=blue size=4>개인자료의 수정-아래는 당신의 자료입니다</font>
<hr>
<center>
<form action=update.jsp?id=<%=id%> method=Post>
<table border=1>
   <tr>
       <td bgcolor=yellow>ID</td><td><%=user%></td>
   </tr>
   <tr>
       <td bgcolor=yellow>성별:</td><td>남<input type=radio name=sex value=남
       <%=session.getValue("boy")%>>녀<input type=radio name=sex value=녀
```

```
        <%=session.getValue("girl")%>></td>
    </tr>
    <tr>
        <td bgcolor=yellow>E-mail</td>
        <td><input type=text size=40 name=email value=<%=email%>></td>
    </tr>
    <tr>
        <td bgcolor=yellow>주소</td>
        <td><input type=text size=60 name=address value=<%=address%>></td>
    </tr>
    <tr>
        <td bgcolor=yellow>통과암호입력</td>
        <td><input type=text size=10 name=password value=<%=password%>></td>
    </tr>
    <tr>
        <td valign=top bgcolor=yellow>다시확인</td>
        <td><input type=password size=10 name=confirm></td>
    </tr>
    <tr align=center><td colspan=2>
        <input type=submit name=send value=수정>
    </tr>
</table>
</form>
<hr>
</BODY>
</HTML>
```

프로그람설명

① 제일 앞부분의 프로그람코드는 자료기지로부터 사용자가 입력한 《이름》과 같은 자료를 추출하며 다음에 자료의 검증과 통과암호의 대비를 진행한다. 만일 오유가 있으면 이전 등록화면에로 돌아와 오유정보를 현시한다.

② HTML원천코드는 하나의 홈을 만들며 매 마당은 이 자료의 값을 추출한다. 실례로 Email란을 《<input type=text name=email value<%=email%>>》으로 설정하면 이 란의 값을 《email》로 하여 자료기지의 email마당값을 설정한다. 만들어진 사용자홈은 그림 8-16에서 보여준다.
사용자는 이 홈에서 직접 자료를 수정할수 있으며 확인통과암호를 다시 입력한 다음 《수정》단추를 찰칵하여 이 홈의 자료를 아래의 update.jsp에 보내 처리할수 있다.

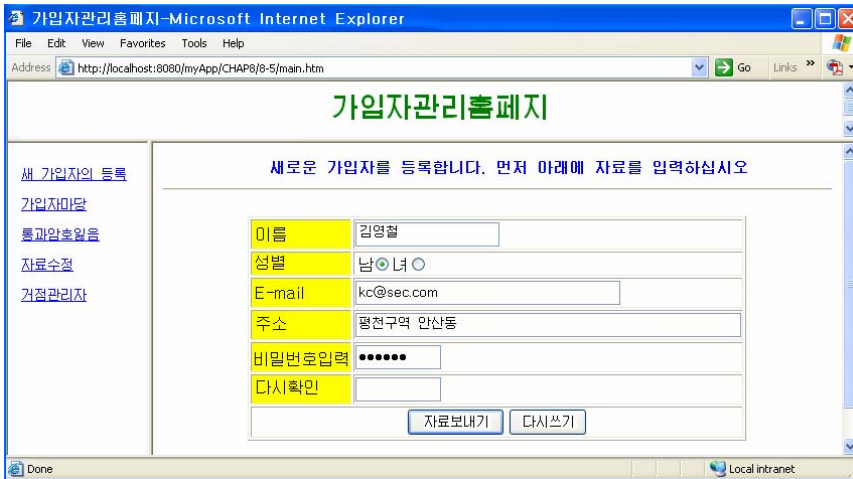



그림 8-16. 가입자관리를 위한 홈페지화면

실례 8-35

update.jsp

```
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%@include file="convert.jsp"%>
<%!
  boolean checkemail(String str)
  {
        int index;
        boolean check;
        if((index=str.indexOf("@"))!=-1)
                check=true;
        else
                check=false;
        return(check);
  }
%>
<%
String id=request.getParameter("id");
String address=request.getParameter("address");
String email=request.getParameter("email");
boolean check=checkemail(email);
String password=request.getParameter("password");
String confirm=request.getParameter("confirm");
```

```
sql="select * from personal where id='"+id+"'";
rs=smt.executeQuery(sql);
out.print("<center>");
 if(check==false)
{
        String errmsg="전자우편주소오유!";
        out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else if(address.length()==0)
{
        String errmsg="주소마당은 빌수 없습니다!";
        out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else if(password.length()>10||password.length()<6)
{
        String errmsg="6~10개 문자의 통과암호를 입력하여야 합니다!";
        out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else if(!password.equals(confirm))
{
        String errmsg="통과암호를 다시 확인하십시오!";
        out.print("<font color=green size=5>오유정보<hr></font><font
color=red>"+errmsg+"</font><hr>");
}
else
{
        String errmsg="당신은 개인자료의 수정을 완성하였습니다!";
        address=convert(address);
        sql="update personal set
email='"+email+"',address='"+address+"',password='"+password+"',where
id='"+id+"'";
        smt.executeQuery(sql);
        out.print("<center><font color=green size=5>"+errmsg+"</font><hr>");
}
out.print("<input type=button value=이전페지로 가기 onclick=history.back();>");
%>
```

프로그람설명

우선 사용자에 의하여 수정된 매 마당의 자료를 얻어 입력이 정확한가를 순차적으로 검사한다. 만일 자료가 틀리면 SQL명령《sql= "update personal set email= ' "+email+" ' ,address= ' "+address+" ' , password= ' "+password+" ' ,where id= ' "+id+" ' ;》을 집행하여 이 자료의 매개 마당내용을 갱신한다.

실례 8-36

manager.jsp

```
<HTML>
<BODY>
<%@page import="java.sql.*"%>
<%@page contentType="text/html;charset=Big5"%>
<%@include file="opendata.jsp"%>
<%
   sql="select * from personal where name='administrator'";
   rs=smt.executeQuery(sql);
   String password=request.getParameter("password");
   String check=rs.getString(4);
   if(session.getValue("enter")==null)
   {
        if(check==null||!password.equals(check))
        {
               String errmsg="err3";
               response.sendRedirect("login.jsp?errmsg="+errmsg+"&login=3");
        }
   }
   session.putValue("enter","true");
   int count=0,lastp,numf,numl,prep,nextp,pageno;
   if(request.getParameter("pageno")==null)
   pageno=0;
   else
   pageno=Integer.parseInt(request.getParameter("pageno"));
   sql="select * from personal";
   rs=smt.executeQuery(sql);
   while(rs.next())
   count++;
   lastp=(int)Math.ceil((double)count/5);
   if(pageno==0||pageno>lastp)
   pageno=lastp;
   numf=pageno*5-4;
```

```
    numl=numf+4;
    if(pageno==1)
    prep=1;
    else
    prep=pageno-1;
    if(pageno==lastp)
    nextp=lastp;
    else
    nextp=pageno+1;
    sql="select * from personal where id between "+numf+"and"+numl;
    rs=smt.executeQuery(sql);
%>
<form action=manager1.jsp method=Post>
<table border=0>
    <tr>
        <td>현재페지번호: <font color=red><%=pageno%></font><font
        color=blue><%=lastp%></font></td>
        <td><a href=manager1.jsp?pageno=<%=prep%>>[이전페지]</a></td>
        <td><a href=manager1.jsp?pageno=<%=nextp%>>[다음페지]</a></td>
        <td><a href=manager1.jsp?pageno=1>[첫페지]</a></td>
        <td><a href=manager1.jsp>[마지막페지]</a></td><td>페지번호입력: <input
        type=text size=3 name=pageno></td>
        <td><input type=submit name=send value=보내기></td>
    </tr>
</table>
</form>
<form action=delete.jsp?pageno=<%=pageno%> method=Post>
<table border=1>
    <tr bgcolor=yellow>
        <td>등록번호</td><td>이름</td><td>E-mail</td>
        <td>주소</td><td>통과암호</td><td>성별</td>
    </tr>
<%
String user,email,address,sex;
int id;
while(rs.next())
{
        user=rs.getString(1);
        email=rs.getString(2);
        address=rs.getString(3);
        password=rs.getString(4);
        sex=rs.getString(5);
```

```
        id=rs.getInt(6);
        out.print("<tr><td><input type=checkbox name=D"+id+"value=del>No.
                        <font color=red>"+id+"</font></td>");
        out.print("<td>"+user+"</td>");
        out.print("<td>"+email+"</td>");
        out.print("<td>"+address+"</td>");
        out.print("<td>"+password+"</td>");
        out.print("<td>");
        if(sex.equals("boy"))
                out.print("남");
        else
                out.print("녀");
        out.print("</td></tr>");
}
%>
<tr><td colspan=6 align=center><input type=submit name=send value=삭제확정>
<input type=reset value=다시선택></td></tr>
</table>
</form>
</BODY>
</HTML>
```

프로그람설명

① 이 프로그람은 관리자가 자료표에서 가입자자료를 찾아보고 어떤 자료를 취하여 삭제를 진행할수 있게 한다. 2절의 게시판과 같은 페지가르기기교를 리용하여 페지를 갈라 열람을 진행할수 있게 프로그람이 작성되여 있으며 여기서 매 페지의 자료개수는 10로 정하였다.

② 관리자등록번호는 《administrator》이며 자료는 꼭같이 personal자료표에서 만들어진다. 사용자가 거점관리자로 등록할 때 이 자료의 등록번호 및 통과암호와 같은가 같지 않는가를 대비한다.

③ 만일 사용자등록이 성공하면 《session.putValue(“enter”, “true”)》로 사용자의 대화접속에서 《enter》변수를 true로 설정한다.

관리자가 자료들을 리용한 후에 《삭제확정》단추를 찰칵하면 delete.jsp을 집행하여 선택된 자료를 자료표로부터 삭제할수 있다.(그림 8-17)

그림 8-17. 등록된 가입자의 정보관리

실례 8-37

delete.jsp

```
<%@page import="java.sql.*"%>
<%@include file="opendata.jsp"%>
<%
String para,value;
int pageno,numf,numl;
if(request.getParameter("pageno")==null)
pageno=0;
else
pageno=Integer.parseInt(request.getParameter("pageno"));
numf=pageno*5-4;
numl=numf+4;
for(int i=numf;i<=numl;i++)
{
    para="D"+String.valueOf(i);
    value=request.getParameter(para);
    if(value!=null)
        {
                sql="delete from personal where id="+i;
                smt.executeQuery(sql);
        }
}
sql="alter table personal drop id"
smt.executeQuery(sql);
```

```
sql="alter table personal add id int auto_increment primary key";
smt.executeQuery(sql);
response.sendRedirect("manager.jsp");
%>
```

프로그람설명

이 프로그람은 관리자가 선택한 자료를 삭제하며 다음에 기본색인란 《id》를 삭제하고 다시 만든다. 이와 같이 자료표의 자료를 다시 번호순서대로 놓는다. 계속하여 《response.sendRedirect(“manager.jsp”)》를 통하여 관리자의 사용화면에로 돌아간다.

JSP와 Servlet

집 필	강 철	심 사	김수홍
편 집	차현옥	교 정	서금석
장 정	서경애	콤퓨터편성	여은정

낸 곳	교육성 교육정보쎈터	인쇄소	교육성 교육정보쎈터
인 쇄	주체97(2008)년 8월 10일	발 행	주체97(2008)년 8월 20일

교-07-1310